EnsEMBL Compara Perl API Tutorial

By Cara Woodwark and Abel Ureta-Vidal

Revisions: CW Jun 04, AUV Aug 04, Nov 04

WARNING: this is still a first draft. A polished version is planned in the near future. By now this tutorial should work with the cvs main trunk of the code (that will soon be branch-ensembl-26), and with ensembl databases release 26 (that will be released for November 2004).

Introduction

This tutorial is an introduction to the ensembl compara API. A knowledge of the ensembl core API is presumed, it is assumed that concepts and conventions presented in the ensembl core API tutorial have been assimilated by the user. The ensembl core API tutorial can be found at http://www.ensembl.org/Docs/linked_docs/ensembl_tutorial.pdf (in cvs, in ensembl/docs/tutorial/ensembl_tutorial.pdf) and should be read first as it provides a comprehensive guide to the ensembl environment.

A documentation about the compara database schema is available http://www.ensembl.org/???? (in cvs ensembl-compara/docs/docs/schema_doc.html), and while not necessary for this tutorial, an understanding of the database tables may help, as many of the Adaptor modules are table specific.

Obtaining the code

To use the ensembl compara API, you have the same requirement that when using the ensembl core API i.e. perl 5.6 or later, bioperl 1.2 or later, DBI, DBD::mysql and ensembl core code. Please refer to the ensembl core API tutorial that will tell you everything about these modules, how and where to get them.

In addition, you will need the ensembl compara code that is available by cvs from the ensembl cvs repository using the following cvs commands:

```
cvs –d :pserver:cvsuser@cvsro.sanger.ac.uk:/cvsroot/CVSmaster login
```

When prompted the password is ‘CVS’.

```
cvs –d :pserver:cvsuser@cvsro.sanger.ac.uk:/cvsroot/CVSmaster co –r branch-ensembl-26
ensembl-compara
```

This will check out ensembl-compara code for stable branch 26. Make sure the ensembl core code you have already checked out is on the same branch. Note that the branch that is checked out should correspond to the database version being used. Thus ensembl_compara_26_1 and e.g. homo-sapiens_core_26_35 and mus_musculus_core_26_33b should be used with the above ensembl branch 26 code.

Environment Variables

The following PERL5LIB environment variables should be set up:

under tcsh/csh shell with

```
setenv PERL5LIB ${PERL5LIB}:{HOME}/src/bioperl-live: \
${HOME}/src/ensembl/modules:${HOME}/src/ensembl-compara/modules
```

under bash shell with

```
export PERL5LIB=${PERL5LIB}:{HOME}/src/bioperl-live: \
${HOME}/src/ensembl/modules:${HOME}/src/ensembl-compara/modules
```

These presume that bioperl and ensembl are in a directory called src set up in your home directory.

Code Conventions (and unconventions)

Refer to the ensembl core tutorial for a good description of the coding conventions normally used in ensembl. Due to historical accidents, there may be exceptions to these rules in compara.

Connecting a ensembl compara database

Explicitely, using the Bio::EnsEMBL::Compara::DBSQL::DBAdaptor

Ensembl compara data as ensembl core data, is stored in a MySQL relational database. If you want to access a compara database, you will need to connect to it. This is done in exactly the same way as when connecting an ensembl core database, but using a Compara specific DBAdaptor.

```
use Bio::EnsEMBL::Compara::DBSQL::DBAdaptor

my $host = ‘ensembldb.ensembl.org’;
my $user = ‘anonymous’;
my $dbname = ‘ensembl_compara_26_1’;

my $comparadb= new Bio::EnsEMBL::Compara::DBSQL::DBAdaptor (-host   => $host,
                                                            -user   => $user,
                                                            -dbname => $dbname);
```

As for a ensembl core connection, in addition to the parameters provided above, the optional `port`, `driver` and `pass` parameters can also be used to specify the TCP connection port, the type of database driver and the password respectively. These values have sensible defaults and can often be omitted.

Implicitely, using the Bio::EnsEMBL::Registry configuration file (recommended)

You will need to have a registry configuration file set up. An example of such file can be found in ensembl/modules/Bio/EnsEMBL/Utils/ensembl_init.example, and below you have a slightly modified copy of it. By default, this file is named .ensembl_init and should be in your home directory.

```
# Example of configuration file used by Bio::EnsEMBL::Registry::load_all method
# to store/register all kind of Adaptors.

use strict;
use Bio::EnsEMBL::Utils::ConfigRegistry;
use Bio::EnsEMBL::DBSQL::DBAdaptor;
use Bio::EnsEMBL::Compara::DBSQL::DBAdaptor;

my @aliases;

new Bio::EnsEMBL::DBSQL::DBAdaptor(-host => 'ensembldb.ensembl.org',
                                   -user => 'anonymous',
                                   -port => 3306,
                                   -species => 'Homo sapiens',
                                   -group => 'core',
                                   -dbname => 'homo_sapiens_core_26_35');

@aliases = ('H_Sapiens', 'homo sapiens', 'Homo_Sapiens','Homo_sapiens', 'Homo',
'homo', 'human');

Bio::EnsEMBL::Utils::ConfigRegistry->add_alias(-species => "Homo sapiens",
                                               -alias => \@aliases);


new Bio::EnsEMBL::DBSQL::DBAdaptor(-host => 'ensembldb.ensembl.org',
                                   -user => 'anonymous',
                                   -port => 3306,
                                   -species => 'Mus musculus',
                                   -group => 'core',
                                   -dbname => 'mus_musculus_core_26_33b');

@aliases = ('M_Musculus', 'mus musculus', 'Mus_Musculus','Mus_musculus', 'Mus', 'mus',
'mouse');

Bio::EnsEMBL::Utils::ConfigRegistry->add_alias(-species => "Mus musculus",
                                               -alias => \@aliases);

new Bio::EnsEMBL::DBSQL::DBAdaptor(-host => 'ensembldb.ensembl.org',
                                   -user => 'anonymous',
                                   -port => 3306,
                                   -species => 'Fugu rubripes',
                                   -group => 'core',
                                   -dbname => 'fugu_rubripes_core_26_2c');

@aliases = ('F_Rubripes', 'fugu rubripes', 'Fugu_Rubripes','Fugu_rubripes', 'Fugu',
'fugu');

Bio::EnsEMBL::Utils::ConfigRegistry->add_alias(-species => "Fugu rubripes",
                                               -alias => \@aliases);

new Bio::EnsEMBL::Compara::DBSQL::DBAdaptor(-host => 'ensembldb.ensembl.org',
                                            -user => 'anonymous',
                                             -port => 3306,
                                             -species => 'Compara26',
                                             -dbname => 'ensembl_compara_26_1');

@aliases = ('ensembl_compara_26_1', 'compara26');

Bio::EnsEMBL::Utils::ConfigRegistry->add_alias(-species => "Compara26",
                                                -alias => \@aliases);

1;
```

In this configuration file, you can list all the parameters needed to connect a compara database. The compara database is a multi-species database that contains comparative genomic information on all ensembl species. One should then be able not only to connect to a compara database but also to every species ensembl core database. The use of the registry configuration file lets you the freedom to list connection parameters for all ensembl core databases you might need to access in relation to ensembl compara data (in our example, only 3 are mentioned, human, mouse and fugu). All this information is then in a single central place, easy to maintain (modify and update).

The access to database adaptor is done using either the main species alias (specified by the `-species` parameter) or one of the aliases specified (in the `@aliases` array). No need to remember the complete database name, one of the aliases will be enough.

WARNING: In previous version of this tutorial, an additional parameter `disconnect_when_active => 1` was specified for all ensembl core databases. It is not needed anymore, as there is now a lazy connection in place i.e. connection will be established only at your first `prepare` statement and kept alive until you use a `disconnect_if_idle` (or a more `disconnect` drastic). If you want to use `disconnect_when_active` make sure you know what you are doing.

Below is a non exhaustive list of ensembl compara adaptors that are most often used

`GenomeDBAdaptor`	to fetch `Bio::EnsEMBL::Compara::GenomeDB` objects
`DnaFragAdaptor`	to fetch `Bio::EnsEMBL::Compara::DnaFrag` objects
`GenomicAlignBlockAdaptor`	to fetch `Bio::EnsEMBL::Compara::GenomicAlignBlock` objects
`DnaAlignFeatureAdaptor`	to fetch `Bio::EnsEMBL::DnaDnaAlignFeature` objects (note that this adaptor return a ensembl core object)
`SyntenyAdaptor`	to fetch `Bio::EnsEMBL::Compara::SyntenyRegion` objects
`MemberAdaptor`	to fetch `Bio::EnsEMBL::Compara::Member` objects
`HomologyAdaptor`	to fetch `Bio::EnsEMBL::Compara::Homology` objects
`FamilyAdaptor`	to fetch `Bio::EnsEMBL::Compara::Family` objects
`PeptideAlignFeatureAdaptor`	to fetch `Bio::EnsEMBL::Compara::PeptideAlignFeature` objects

Only some of these adaptors will be used for illustration as part of this tutorial through commented perl scripts code.

Whole Genome Alignments

The compara database contains a number of different types of whole genome alignments. A much detailed information about what are these different types can be found here

```
http://www.ensembl.org/Homo_sapiens/helpview?se=1&kw=contigview#mus_musculus_match
```

The full range of comparisons available may be found in the method_link table as shown below.

```
mysql> select * from method_link;
+----------------+----------------------+
| method_link_id | type                 |
+----------------+----------------------+
|              1 | BLASTZ_NET           |
|              2 | BLASTZ_NET_TIGHT     |
|              3 | BLASTZ_RECIP_NET     |
|              4 | PHUSION_BLASTN       |
|              5 | PHUSION_BLASTN_TIGHT |
|              6 | TRANSLATED_BLAT      |
|              7 | BLASTZ_GROUP         |
|              8 | BLASTZ_GROUP_TIGHT   |
|            101 | SYNTENY              |
|            201 | ENSEMBL_ORTHOLOGUES  |
|            301 | FAMILY               |
|            202 | ENSEMBL_PARALOGUES   |
+----------------+----------------------+
12 rows in set (0.00 sec)
```

The designation ‘TIGHT’ denotes that the alignments have been rescored using the ‘TIGHT’ matrix (in cvs, ensembl-compara/scripts/hcr/tight.mat) so only the most highly conserved alignments are reported.

The whole genome comparisons can be accessed through the API by 2 different ways using of the 2 different adaptors. Specifically, the DnaAlignFeatureAdaptor, which returns DnaDnaAlignFeatures objects (only used for pairwise alignment) and the GenomicAlignBlockAdaptor, which returns GenomicAlignBlock objects (can be used for pairwise and also multiple alignments).

DnaDnaAlignFeature objects (for pairwise alignments only)

Below it is a simple commented perl script to illustrate the use of

```
use strict;
use Bio::EnsEMBL::Registry;
use Bio::EnsEMBL::Compara::DBSQL::DBAdaptor;
use Bio::AlignIO;
use Bio::LocatableSeq;
use Getopt::Long;

my $usage = "
$0
  [--help]                    this menu
   --dbname string            (e.g. compara23) one of the compara database
                              Bio::EnsEMBL::Registry aliases
   --seq_region string        (e.g. 22)
   --seq_region_start integer   (e.g. 50000000)
   --seq_region_end integer     (e.g. 50500000)
   --qy string                (e.g. human) the query species (i.e. a
                              Bio::EnsEMBL::Registry alias) from which alignments
                              are queried and seq_region refer to
   --tg string                (e.g. mouse) the target sepcies (i.e. a
                              Bio::EnsEMBL::Registry alias) to which alignments are
                              queried
  [--alignment_type string]     (e.g. TRANSLATED_BLAT) type of alignment stored
                              (default: BLASTZ_NET)
  [--tsl]                       print out a translated alignment
  [--oo]                        By default, the alignments are dumped so that the --qy
                              species sequence is always on forward strand. --oo is
                              mostly useful in association with -tsl option, when a
                              full translated alignment program has been used e.g
                              TRANSLATED_BLAT, and allow to obtain the right
                              translation phase. So the --qy species sequence might
                              be reverse complemented.
  [--ft string]                 alignment format, available in bioperl Bio::AlignIO
                              (default: clustalw)
  [--uc]                        print out sequence in upper cases (default is lower
                              cases)
  [--limit integer]             (e.g. 2) limit the output to the number of alignments
                              specified
  [--reg_conf filepath]         the Bio::EnsEMBL::Registry configuration file. If none
                              given, the one set in ENSEMBL_REGISTRY will be used if
                              defined, if not ~/.ensembl_init will be used.

\n";

my $dbname;
my ($seq_region,$seq_region_start,$seq_region_end);
my ($qy_species,$tg_species);
my $help = 0;
my $alignment_type = "BLASTZ_NET";
my $limit;
my $reg_conf;
my $format = "clustalw";
my $translated = 0;
my $uc = 0;
my $original_orientation = 0;

unless (scalar @ARGV) {
  print $usage;
  exit 0;
}

GetOptions('help' => \$help,
           'dbname=s' => \$dbname,
           'seq_region=s' => \$seq_region,
           'seq_region_start=i' => \$seq_region_start,
           'seq_region_end=i' => \$seq_region_end,
           'qy=s' => \$qy_species,
           'tg=s' => \$tg_species,
           'alignment_type=s' => \$alignment_type,
           'tsl' => \$translated,
           'ft=s' => \$format,
           'uc' => \$uc,
           'oo' => \$original_orientation,
           'limit=i' => \$limit,
           'reg_conf=s' => \$reg_conf);

$|=1;

if ($help) {
  print $usage;
  exit 0;
}

# Setting up Bio::EnsEMBL::Regitry
# if $reg_conf is undef, ~/.ensembl_init will be loaded if it exists

Bio::EnsEMBL::Registry->load_all($reg_conf);

$format = lc $format;

# Getting the core SliceAdaptor for the query species

my $qy_sa = Bio::EnsEMBL::Registry->get_adaptor($qy_species,'core','Slice');

# Fetching a Slice. In compara, all slices are ‘toplevel’ coordinate system.

my $qy_slice = $qy_sa->fetch_by_region('toplevel',$seq_region,
                                       $seq_region_start,$seq_region_end);
```

```
# Getting the core MetaContainer adaptor for the target species

my $tg_mc = Bio::EnsEMBL::Registry->get_adaptor($tg_species,'core','MetaContainer');

# Getting a Bio::Species object and from it the Species genus (e.g. Mus
# musculus) of the target species, using the binomial call

my $tg_binomial = $tg_mc->get_Species->binomial;

# Getting the compara DnaAlignFeatureAdaptor to query the compara database

my $dafad = Bio::EnsEMBL::Registry->get_adaptor($dbname,'compara','DnaAlignFeature');

# Fetching DnaDnaAlignFeatures object (these are core objects) using the
# fetch_all_by_Slice. The 3rd argument that can specify the assembly version
# can be undef. The compara API will find for you the default assembly for
# the target species.

my $DnaDnaAlignFeatures =
$dafad->fetch_all_by_Slice($qy_slice,$tg_binomial,undef,$alignment_type,$limit);

# Go through each alignment to print out in the requested format

foreach my $ddaf (sort {$a->start <=> $b->start
                         || $a->end <=> $b->end}
                  @{$DnaDnaAlignFeatures}) {

  # if the original alignment strand orientation is requested
  # ($original_orientation is true) and effectively the alignment obtained
  # is reverse complement from the originally obtained by the alignment
  # program used (if $ddaf->strands_reversed is true), then reverse
  # complement the alignment.

  if ($original_orientation && $ddaf->strands_reversed) {
    $ddaf->reverse_complement;
  }

  # Create a list of flags to be used in the get_SimpleAlign method call

  my @flags;
  push @flags, 'translated' if ($translated);
  push @flags, 'uc' if ($uc);

  # Get a Bio::SimpleAlign from the DnaDnaAlignFeature object

  my $sa = $ddaf->get_SimpleAlign(@flags);

  # Create a Bio::AlignIO with the requested output format

  my $alignIO = Bio::AlignIO->newFh(-interleaved => 0,
                                    -fh => \*STDOUT,
                                    -format => $format,
                                    -idlength => 20);

  # print out the alignment (Bio::SimpleAlign object) in the requested
  # output format through the Bio::AlignIO handler

  print $alignIO $sa;
}

exit 0;
```

So to pull out BLASTZ_NET_TIGHT alignments, let’s say on part of ENCODE region ENm004 on human chromosome 22, between position 30184430 and position 30184485, against the mouse genome in clustalw format, we can use know the following command line,

```
% perl DumpAlignmentsLight.pl --dbname Compara23 --seq_region 22
```

```
--seq_region_start 30184430 --seq_region_end 30184485 --qy human --tg mouse
--alignment_type BLASTZ_NET_TIGHT

CLUSTAL W(1.81) multiple sequence alignment


22/30184223-30184547   tgaaacgcttgtccttgaagtccctctctcggtctctgtctctcaagtcccgcaggtcct
11/3118113-3118437     tgaaacgtttgtccttgtagtccctctctctgtctcggtctctcaagtctcgcaggtcct
                       ******* ********* ************ ***** ************ **********


22/30184223-30184547   tatcgctaagacggtgatccttctcaaaggtccgggcagagattatcctcccactgccaa
11/3118113-3118437     tatcactgagacggtgatcctttttcaaaggcccgggcagaaattatccttccactgccaa
                       **** ** ************** ******* ********* ******** **********


22/30184223-30184547   tcctacgtccaccaagcagacgaagcccatcactatctttctctaatggacttcctgagc
11/3118113-3118437     ttcttcgtccaccaagcaggcgaagtccatcactgtctttctccaatggactgccagatc
                       * ** ************** ***** ******** ******** ******** ** ** *


22/30184223-30184547   gccgggagctaacagcggctgtcacgtggcagccccctccaaagctccgtctctgagggc
11/3118113-3118437     gtcgggagctaacagcagctgtcacatggcagccacctccaaagcttcgtctctgtgggc
                       * ************** ******** ******** *********** ******** ****


22/30184223-30184547   tgagaacaacatctaagtcatcttctttcacacgctctcgtggatctggaaggacgtggg
11/3118113-3118437     tgagaacaacatctaagtcatcttctttcactcgctctcgtggatctgaaaagatgccag
                       ******************************* **************** ** ** *   *


22/30184223-30184547   aaagacaaagttaaacaaaccaaca
11/3118113-3118437     aaagagaaaggtaagcaaaccaaca
                       ***** **** *** **********
```

Now on the same region, TRANSLATED_BLAT alignments against fugu in clustalw format, but at translation level now (-tsl) not nucleotide level, we can run the following command line,

```
% perl DumpAlignments.pl --dbname Compara26 --seq_region 22 \
--seq_region_start 30184430 --seq_region_end 30184485 --qy human --tg fugu \
--alignment_type TRANSLATED_BLAT --tsl

CLUSTAL W(1.81) multiple sequence alignment


22/30184431-30184484    aapskapslraenni*vi
scaffold_2267/1347-1400 tspskaaplwa*yyi*ii
                        ::****..* *   **:*
```

By default, the alignments will dump with --qy species sequence on forward strand. To make sure that the alignment, you got is on the strand on which it was originally generated using the --oo option will check that and restore the right strandness. See below the difference in the translation level alignment obtained.

```
% perl DumpAlignments.pl --dbname Compara26 --seq_region 22 \
--seq_region_start 30184430 --seq_region_end 30184485 --qy human --tg fugu \
--alignment_type TRANSLATED_BLAT --tsl --oo

CLUSTAL W(1.81) multiple sequence alignment


22/30184431-30184484    ddldvvlspqrrsfgggc
scaffold_2267/1347-1400 ddldvilspqrrsfgggc
                        *****:************
```

GenomicAlignBlock objects (pairwise/multiple alignments)

To be written.

Orthologues and Protein clusters

NB : This following is very much a draft at this stage with some piece of code to give examples, but not much comments.

Member objects

```
# get the MemberAdaptor
my $ma = Bio::EnsEMBL::Registry->get_adaptor($dbname,'compara','Member');
# fetch a Member
my $member = $ma->fetch_by_source_stable_id('ENSEMBLGENE','ENSG00000004059');
# print out some information about the Member
print join " ", map { $member->$_ } qw(chr_name chr_start chr_end description
source_name taxon_id taxon),"\n";
```

chr_name, chr_start, chr_end and description are self-explanatory.

source_name tells about the origin of the Member entry, and can be either

ENSEMBLPEP, derived from ensembl translation,

or ENSEMBLGENE, derived from an ensembl gene,

or SWISSPROT, derived from a Uniprot/Swissprot entry,

or SPTREMBL, derived from a Uniprot/SP-TrEMBL entry.

taxon_id e.g. 9606 correspond to the NCBI taxonomy identifier (see http://www.ncbi.nlm.nih.gov/Taxonomy/taxonomyhome.html/ for more details).

taxon returns a Bio::EnsEMBL::Compara::Taxon object that inherits itself from Bio::Species, so from this object you can get additional information about the species.

```
my $taxon = $member->taxon;
print join "; ", map { $taxon->$_ } qw(common_name genus species binomial
classification),"\n";
```

respectively for these method calls and in the case of human species, you will obtain

```
human; Homo; sapiens; Homo sapiens; sapiens Homo Hominidae Catarrhini Primates
Eutheria Mammalia Euteleostomi Vertebrata Craniata Chordata Metazoa Eukaryota
```

Homology objects

```
# first you have to get a Member object. In case of homology is a gene, in
# case of family it can be a gene or a protein

my $ma = Bio::EnsEMBL::Registry->get_adaptor($dbname,'compara','Member');
my $member = $ma->fetch_by_source_stable_id('ENSEMBLGENE','ENSG00000004059');

# then you get the homologies where the member is involved

my $ha = Bio::EnsEMBL::Registry->get_adaptor($dbname,'compara','Homology');
my $homologies = $ha->fetch_by_Member($member);
fetch_by_Member_Homology_source (fetch_by_Member_MethodLink)

# That will return an array reference with all homologies (orthologues, and
# in some cases paralogues) against other species.
# Then for each homology, you get all the Members implicated

foreach my $homology (@{$homologies}) {
```

```
# You will find different kind of description
# UBRH, MBRH, MBRH, RHS, YoungParalogues
# see ensembl-compara/docs/docs/schema_doc.html for more details

print $homology->description,” ”, $homology->subtype,”\n”;

# And if they are defined dN and dS related values

print join “ “, map { $homology->$_ } qw(dn ds n s lnl threshold_on_ds),”\n”;

# each homology relation have only 2 members, you should find there
# the initial member used in the first fetching

for each my $member_attribute (@{$homology->get_all_Member_Attribute})

# for each Member, you get information on the Member specifically and in
# relation to the homology relation via Attribute object

    my ($member, $attribute) = @{$member_attribute};
    print join “ “, map { $member->$_ }  qw(stable_id taxon_id),"\n";
    print join “ “, map { $attribute->$_ } qw(perc_id perc_pos perc_cov),"\n";

  }

# You can even retrieve the HSP alignment between the 2 proteins,
# HSP used to build the homology releationship at the peptide level

  my $sa = $homology->get_SimpleAlign();
  my $alignIO = Bio::AlignIO->newFh(-interleaved => 0,
                                    -fh => \*STDOUT,
                                    -format => "clustalw",
                                    -idlength => 20);

  print $alignIO $sa;

# or at the nucleotide level. You will need to make you have a connection to
# the corresponding core databases through the Bio::EnsEMBL::Registry

  $sa = $homology->get_SimpleAlign(‘cdna’);
  my $alignIO = Bio::AlignIO->newFh(-interleaved => 0,
                                    -fh => \*STDOUT,
                                    -format => "phylip",
                                    -idlength => 20);

  print $alignIO $sa;

}
```

Family objects

```
my $ma = Bio::EnsEMBL::Registry->get_adaptor($dbname,’compara’,’Member’);
my $member = $ma->fetch_by_source_stable_id(‘ENSEMBLGENE’,’ENSG00000004059’);

my $fa = Bio::EnsEMBL::Registry->get_adaptor($dbname,’compara’,’Family’);
my $families = $fa->fetch_by_Member($member);

foreach my $family (@{$families}) {
 print $family->description;

 for each my $member_attribute (@{$family->get_all_Member_Attribute})
    my ($member, $attribute) = @{$member_attribute};
    print $member->stable_id," ",$member->taxon_id,"\n";
 }

  my $sa = $family->get_SimpleAlign();
  my $alignIO = Bio::AlignIO->newFh(-interleaved => 0,
                                    -fh => \*STDOUT,
                                    -format => "phylip",
                                    -idlength => 20);

  print $alignIO $sa;

  $sa = $family->get_SimpleAlign(‘cdna’);
  my $alignIO = Bio::AlignIO->newFh(-interleaved => 0,
                                      -fh => \*STDOUT,
                                      -format => "phylip",
                                      -idlength => 20);

  print $alignIO $sa;
}
```