EnsEMBL Compara Perl API Tutorial

By Cara Woodwark, Abel Ureta-Vidal and Javier Herrero

Revisions: CW Jun 04, AUV Aug 04, Nov 04, JH Nov 04, Apr 05, BB Jan 07, LG Nov 08

NB: this tutorial has been tested to work with branch-ensembl-51, and with EnsEMBL databases release 51. However you may find still errors in it. Please email ensembl-dev@ebi.ac.uk, so that we can correct them.

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http://www.ensembl.org/info/docs/api/core/core_tutorial.html (and in Nov'2008 a slightly outdated CVS version in ensembl/docs/tutorial/ensembl_tutorial.pdf) and should be read first as it provides a comprehensive guide to the EnsEMBL environment.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 at http://www.ensembl.org/info/docs/api/compara/compara_schema.html and while not absolutely 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.

You may start by creating a directory for storing the API in your home directory:

```
cd
mkdir src
cd src
```

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@cvs.sanger.ac.uk:/cvsroot/ensembl login
```

When prompted the password is "CVSUSER".

```
cvs -d :pserver:cvsuser@cvs.sanger.ac.uk:/cvsroot/ensembl co -r branch-ensembl-51
ensembl-compara
```

This will check out *ensembl-compara* code for stable branch 51. 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_51* and e.g. *homo_sapiens_core_51_36m* and *mus_musculus_core_51_37d* should be used with the above ensembl branch 51 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.

Connecting to an EnsEMBL Compara database

Connection parameters

Starting from release 48 EnsEMBL is running two public MySQL servers on *host=ensembldb.ensembl.org* with two different port numbers. The server on *port=3306* hosts all databases prior to rel.48 and the server on *port=5306* hosts all newer databases starting from rel.48.

There are two API ways to connect to the EnsEMBL Compara database:

- In most cases you will prefer the implicit way - using Bio::EnsEMBL::Registry module, which can read either a global or a specific configuration file or auto-configure itself.
- However there are cases where you might want more flexibility provided by the explicit creation of a Bio::EnsEMBL::Compara::DBSQL::DBAdaptor.

Implicitly, using the Bio::EnsEMBL::Registry auto-configuration feature (recommended)

For using the auto-configuration feature, you will first need to supply the connection parameters to the Registry loader. For instance, if you want to connect to the the public EnsEMBL databases you can use the following command in your scripts:

```
use Bio::EnsEMBL::Registry;
Bio::EnsEMBL::Registry->load_registry_from_db(
        -host => 'ensembldb.ensembl.org',
        -user => 'anonymous',
        -port => 5306
);
```

This will initialize the Registry, from which you will be able to create object-specific adaptors later. Alternatively, you can use a shorter version based on a URL:

```
use Bio::EnsEMBL::Registry;
Bio::EnsEMBL::Registry>load_registry_from_url(
        'mysql://anonymous@ensembldb.ensembl.org:5306/');
```

Implicitly, using the Bio::EnsEMBL::Registry configuration file

You will need to have a registry configuration file set up. By default, it takes the file defined by the ENSEMBL_REGISTRY environment variable or the file named .ensembl_init in your home directory if the former is not found. Additionally, you can use a specific file (see perldoc Bio::EnsEMBL::Registry or later in this document for some examples on how to use a different file). Please, refer to the EnsEMBL Registry documentation (http://www.ensembl.org/info/docs/api/registry.html) for details about this option.

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

EnsEMBL Compara data, like 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 to connect to an EnsEMBL core database, but using a Compara-specific DBAdaptor. One parameter you have to supply in addition to the ones needed by the Registry is the -dbname, which by convention contains the release number:

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

my $host   = 'ensembldb.ensembl.org';
my $user   = 'anonymous';
my $port   = 5306;
my $dbname = 'ensembl_compara_51';

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

EnsEMBL Compara object-specific adaptors

EnsEMBL Compara adaptors are used to fetch data from the database. Data are returned as EnsEMBL objects. For instance, the GenomeDBAdaptor returns Bio::EnsEMBL::Compara::GenomeDB objects.

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 returns an 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.

You can get the adaptors from the Registry with the get_adaptor command. You need to specify three arguments: the species name, the type of database and the type of object. Therefore, in order to get the GenomeDBAdaptor for the Compara database, you will need the following command:

```
my $genome_db_adaptor = Bio::EnsEMBL::Registry->get_adaptor(
    'Multi', 'compara', 'GenomeDB');
```

Note: As the EnsEMBL Compara DB is a multi-species database, the standard species name is 'Multi'. The type of the database is 'compara'.

Code Conventions

Refer to the EnsEMBL core tutorial (http://www.ensembl.org/info/docs/api/core/index.html) for a good description of the coding conventions normally used in EnsEMBL.

We can divide the fetching methods of the ObjectAdaptors into two categories: the *fetch_by* and *fetch_all_by*. The former return one single object while the latter return a reference to an array of objects.

```
my $this_genome_db = $genome_db_adaptor
                       ->fetch_by_name_assembly("Homo sapiens", "NCBI36");
my $all_genome_dbs = $genome_db_adaptor->fetch_all();
foreach my $this_genome_db (@{$all_genome_dbs}) {
  print $this_genome_db->name, "\n";
}
```

Whole Genome Alignments

The Compara database contains a number of different types of whole genome alignments. A listing about what are these different types can be found in the ensembl-compara/docs/schema_doc.html document in method_link section.

GenomicAlignBlock objects (pairwise/multiple alignments)

GenomicAlignBlocks are the preferred way to store and fetch genomic alignments. A GenomicAlignBlock contains several GenomicAlign objects. Every GenomicAlign object corresponds to a piece of genomic sequence aligned with the other GenomicAlign in the same GenomicAlignBlock. A GenomicAlign object is always related with other GenomicAlign objects and this relation is defined through the GenomicAlignBlock object. Therefore the usual way to fetch genomic alignments is by fetching GenomicAlignBlock objects. We have to start by getting the corresponding adaptor:

```
# Getting the GenomicAlignBlock adaptor:
my $genomic_align_block_adaptor = Bio::EnsEMBL::Registry->get_adaptor(
        'Multi, 'compara', 'GenomicAlign');
```

In order to fetch the right alignments we need to specify a couple of data: the type of alignment and the piece of genomic sequence in which we are looking for alignments. The type of alignment is a more tricky now: you need to specify both the alignment method and the set of genomes. In order to simply this task, you could use the new `Bio::EnsEMBL::Compara::MethodLinkSpeciesSet` object. The best way to use them is by fetching them from the database:

```
# Getting the GenomeDB adaptor:
my $genome_db_adaptor = Bio::EnsEMBL::Registry->get_adaptor(
        $dbname, 'compara', 'GenomeDB');
# Fetching GenomeDB objects for human and mouse:
my $human_genome_db = $genome_db_adaptor->fetch_by_name_asembly('Homo sapiens');
my $mouse_genome_db = $genome_db_adaptor->fetch_by_name_asembly('Homo sapiens');
# Getting the MethodLinkSpeciesSet adaptor:
my $method_link_species_set_adaptor = Bio::EnsEMBL::Registry->get_adaptor(
        $dbname, 'compara', 'MethodLinkSpeciesSet');
# Fetching the MethodLinkSpeciesSet object corresponding to BLASTZ_NET
alignments between human and mouse genomic sequences:
my $human_mouse_blastz_net_mlss =
        $method_link_species_set_adaptor->fetch_by_method_link_type_GenomeDBs(
                'BLASTZ_NET',
                [$human_genome_db, $mouse_genome_db]
        );
```

There are two ways to fetch GenomicAlignBlocks. One is uses `Bio::EnsEMBL::Slice` objects while the second one is based on `Bio::EnsEMBL::Compara::DnaFrag` objects for specifying the piece of genomic sequence in which we are looking for alignments.

```
my $query_species = 'human';
my $seq_region = '14';
my $seq_region_start = 75000000;
my $seq_region_end   = 75010000;

# Getting the Slice adaptor:
my $slice_adaptor = Bio::EnsEMBL::Registry->get_adaptor(
        $query_species, 'core', 'Slice');
# Fetching a Slice object:
my $query_slice = $qy_sa->fetch_by_region('toplevel', $seq_region, $seq_region_start,
        $seq_region_end);
# Fetching all the GenomicAlignBlock corresponding to this Slice:
my $genomic_align_blocks =
        $genomic_align_block_adaptor->fetch_by_MethodLinkSpeciesSet_Slice(
                $human_mouse_blastz_net_mlss, $query_slice);
```

Here is an example script with all of this:

```
use strict;
use Bio::EnsEMBL::Registry;
use Bio::EnsEMBL::Utils::Exception qw(throw);
use Bio::SimpleAlign;
use Bio::AlignIO;
use Bio::LocatableSeq;
use Getopt::Long;

my $usage = qq{
perl DumpMultiAlign.pl
  Getting help:
    [--help]

  General configuration:
    [--reg_conf registry_configuration_file]
        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.
    [--dbname compara_db_name]
        the name of Compara DB in the registry_configuration_file or any
        of its aliases. Uses "compara" by default.

  For the query slice:
    [--species species]
        Query species. Default is "human"
    [--coord_system coordinates_name]
        Query coordinate system. Default is "chromosome"
    --seq_region region_name
        Query region name, i.e. the chromosome name
    --seq_region_start start
    --seq_region_end end

  For the alignments:
    [--alignment_type method_link_name]
        The type of alignment. Default is "BLASTZ_NET"
    [--set_of_species species1:species2:species3:...]
        The list of species used to get those alignments. Default is
        "human:mouse". The names should correspond to the name of the
        core database in the registry_configuration_file or any of its
        aliases

  Ouput:
    [--output_format clustalw|fasta|...]
        The type of output you want. "clustalw" is the default.
    [--output_file filename]
        The name of the output file. By default the output is the
        standard output
};

my $reg_conf;
my $dbname = "compara";
my $species = "human";
my $coord_system = "chromosome";
my $seq_region = "14";
my $seq_region_start = 75000000;
```

```
my $seq_region_end = 75010000;
my $alignment_type = "BLASTZ_NET";
my $set_of_species = "human:mouse";
my $output_file = undef;
my $output_format = "clustalw";
my $help;

GetOptions(
    "help" => \$help,
    "reg_conf=s" => \$reg_conf,
    "dbname=s" => \$dbname,
    "species=s" => \$species,
    "coord_system=s" => \$coord_system,
    "seq_region=s" => \$seq_region,
    "seq_region_start=i" => \$seq_region_start,
    "seq_region_end=i" => \$seq_region_end,
    "alignment_type=s" => \$alignment_type,
    "set_of_species=s" => \$set_of_species,
    "output_format=s" => \$output_format,
    "output_file=s" => \$output_file,
  );

# Print Help and exit
if ($help) {
  print $usage;
  exit(0);
}

if ($output_file) {
  open(STDOUT, ">$output_file") or die("Cannot open $output_file");
}

# Configure the Bio::EnsEMBL::Registry
# Uses $reg_conf if supllied. Uses ENV{ENSMEBL_REGISTRY} instead if defined.
# Uses ~/.ensembl_init if all the previous fail.
Bio::EnsEMBL::Registry->load_all($reg_conf);

# Getting all the Bio::EnsEMBL::Compara::GenomeDB objects
my $genome_dbs;
my $genome_db_adaptor = Bio::EnsEMBL::Registry->get_adaptor($dbname, 'compara',
        'GenomeDB');
throw("Registry configuration file has no data for connecting to <$dbname>")
        if (!$genome_db_adaptor);
foreach my $this_species (split(":", $set_of_species)) {
  my $this_meta_container_adaptor = Bio::EnsEMBL::Registry->get_adaptor(
      $this_species, 'core', 'MetaContainer');
  throw("Registry configuration file has no data for connecting to <$this_species>")
      if (!$this_meta_container_adaptor);
  my $this_binomial_id = $this_meta_container_adaptor->get_Species->binomial;
  # Fetch Bio::EnsEMBL::Compara::GenomeDB object
  my $genome_db = $genome_db_adaptor->fetch_by_name_assembly($this_binomial_id);
  # Add Bio::EnsEMBL::Compara::GenomeDB object to the list
  push(@$genome_dbs, $genome_db);
}

# Getting Bio::EnsEMBL::Compara::MethodLinkSpeciesSet obejct
my $method_link_species_set_adaptor = Bio::EnsEMBL::Registry->get_adaptor(
        $dbname, 'compara', 'MethodLinkSpeciesSet');
my $method_link_species_set =
        $method_link_species_set_adaptor->fetch_by_method_link_type_GenomeDBs(
                $alignment_type, $genome_dbs);
throw("The database do not contain any $alignment_type data for $set_of_species!")
        if (!$method_link_species_set);

# Fetching the query Slice:
my $slice_adaptor = Bio::EnsEMBL::Registry->get_adaptor($species, 'core', 'Slice');
throw("Registry configuration file has no data for connecting to <$species>")
        if (!$slice_adaptor);
my $query_slice = $slice_adaptor->fetch_by_region('toplevel', $seq_region,
$seq_region_start, $seq_region_end);
throw("No Slice can be created with coordinates $seq_region:$seq_region_start-".
        "$seq_region_end") if (!$query_slice);

# Fetching all the GenomicAlignBlock corresponding to this Slice:
my $genomic_align_block_adaptor = Bio::EnsEMBL::Registry->get_adaptor(
    $dbname, 'compara', 'GenomicAlignBlock');
my $genomic_align_blocks =
    $genomic_align_block_adaptor->fetch_all_by_MethodLinkSpeciesSet_Slice(
        $method_link_species_set, $query_slice);

my $all_aligns;
# Create a Bio::SimpleAlign object from every GenomicAlignBlock
foreach my $this_genomic_align_block (@$genomic_align_blocks) {
  my $simple_align = Bio::SimpleAlign->new();
  $simple_align->id("GAB#".$this_genomic_align_block->dbID);
```

```
  $simple_align->score($this_genomic_align_block->score);

  my $all_genomic_aligns = $this_genomic_align_block->get_all_GenomicAligns;
  # Create a Bio::LocatableSeq object from every GenomicAlign
  foreach my $this_genomic_align (@$all_genomic_aligns) {
    my $seq_name = $this_genomic_align->dnafrag->genome_db->name;
    $seq_name =~ s/(.)\w* (.)\w*/$1$2/;
    $seq_name .= $this_genomic_align->dnafrag->name;
    my $aligned_sequence = $this_genomic_align->aligned_sequence;
    my $seq = Bio::LocatableSeq->new(
            -SEQ    => $aligned_sequence,
            -START  => $this_genomic_align->dnafrag_start,
            -END    => $this_genomic_align->dnafrag_end,
            -ID     => $seq_name,
            -STRAND => $this_genomic_align->dnafrag_strand
        );
    # Add this Bio::LocatableSeq to the Bio::SimpleAlign
    $simple_align->add_seq($seq);
  }
  push(@$all_aligns, $simple_align);
}

# print all the genomic alignments using a Bio::AlignIO object
my $alignIO = Bio::AlignIO->newFh(
        -interleaved => 0,
        -fh => \*STDOUT,
        -format => $output_format,
        -idlength => 10
    );

foreach my $this_align (@$all_aligns) {
  print $alignIO $this_align;
}

exit;
```

Homologies and Protein clusters

All the homologies and families refer to Members. Homology objects store orthologous and paralogous relationships between Members and Family objects are clusters of Members.

Member objects

A Member represent either a gene or a protein. Most of them are defined in the corresponding EnsEMBL core database. For instance, the sequence for the human gene ENSG00000004059 is stored in the human core database.

The *fetch_by_source_stable_id* method of the MemberAdaptor takes two arguments. The first one is the source of the Member and can be:

- ENSEMBLPEP, derived from an EnsEMBL translation
- ENSEMBLGENE, derived from an EnsEMBL gene
- Uniprot/SWISSPROT, derived from a Uniprot/Swissprot entry
- Uniprot/SPTREMBL, derived from a Uniprot/SP-TrEMBL entry

The second argument is the identifier for the Member. Here is a simple example:

```
        # get the MemberAdaptor
        my $member_adaptor = Bio::EnsEMBL::Registry->get_adaptor(
            'Multi','compara','Member');

        # fetch a Member
        my $member = $member_adaptor->fetch_by_source_stable_id(
            'ENSEMBLGENE','ENSG00000004059');

        # print out some information about the Member
        print $member->chr_name, " ( ", $member->chr_start, " - ", $member->chr_end,
            " ): ", $member->description, "\n";
```

The Member object has several attributes:

- *source_name* and *stable_id* define this Member.
- *chr_name*, *chr_start*, *chr_end*, *chr_strand* locate this Member on the genome but are only available for ENSEMBLGENE and ENSEMBLPEP.
- *taxon_id* corresponds to the NCBI taxonomy identifier (see http://www.ncbi.nlm.nih.gov/Taxonomy/taxonomyhome.html/ for more details).
- *taxon* returns a Bio::EnsEMBL::Compara::NCBITaxon object. From this object you can get additional information about the species.

```
my $taxon = $member->taxon;
print "common_name ", $taxon->common_name,"\n";
print "genus ", $taxon->genus,"\n";
print "species ", $taxon->species,"\n";
print "binomial ", $taxon->binomial,"\n";
print "classification ", $taxon->classification,"\n";
```

In our example the species is human, so the output will look like this:

```
common_name:    human

genus:          Homo

species:        sapiens

binomial:       Homo sapiens

classification:    sapiens    Homo    Hominidae    Catarrhini    Haplorrhini    Primates
Euarchontoglires Eutheria Mammalia Euteleostomi Vertebrata Craniata Chordata Metazoa
Eukaryota
```

Homology Objects

A Homology object represents either an orthologous or paralogous relationships between two or more Members.

Typically you want to get homologies for a given gene. The HomologyAdaptor has a fetching method called fetch_all_by_Member(). You will need the Member object for your query gene, therefore you will fetch the Member first like in this example:

```
# 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 $member_adaptor = Bio::EnsEMBL::Registry
                ->get_adaptor('Multi', 'compara', 'Member');
my $member = $member_adaptor
                ->fetch_by_source_stable_id('ENSEMBLGENE','ENSG00000004059');

# then you get the homologies where the member is involved

my $homology_adaptor = Bio::EnsEMBL::Registry
                ->get_adaptor('Multi', 'compara', 'Homology');
my $homologies = $homology_adaptor->fetch_all_by_Member($member);

# That will return a reference to an array with all homologies (orthologues in
# other species and paralogues in the same one)
# Then for each homology, you can get all the Members implicated

foreach my $homology (@{$homologies}) {
  # You will find different kind of description
  # UBRH, 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 " dn ", $homology->dn,"\n";
  print " ds ", $homology->ds,"\n";
  print " dnds_ratio ", $homology->dnds_ratio,"\n";
}
```

Each homology relation has 2 or more members, you should find there the initial member used as a query. The get_all_MemberAttribute method returns an array of pairs of Member and Attributes. The Member corresponds to the gene or protein and the Attribute object contains information about how this Member has been aligned.

```
my $homology = $homologies->[0]; # take one of the homologies and look into it

foreach 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 get the original alignment used to define an homology:

```
        use Bio::AlignIO;

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

        print $alignIO $simple_align;
```

Family Objects

Families are clusters of proteins including all the EnsEMBL proteins plus all the metazoan SwissProt and SP-Trembl entries. The object and the adaptor are really similar to the previous ones.

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

my $family_adaptor = Bio::EnsEMBL::Registry->get_adaptor('Multi','compara','Family');
my $families = $family_adaptor->fetch_all_by_Member($member);

foreach my $family (@{$families}) {
    print join(" ", map { $family->$_ }  qw(description description_score))."\n";

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

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

    print $alignIO $simple_align;

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

    print $alignIO $simple_align;
}
```

Further help

For additional information or help mail the ensemb-dev@ebi.ac.uk mailing list. You will need to subscribe to this mailing list to use it (see how to subscribe in http://www.ensembl.org/info/about/contact/mailing.html).