Introduction to the Ensembl REST API

EMBL-EBI

e!Ensembl
## This webinar course

<table>
<thead>
<tr>
<th>Date</th>
<th>Webinar topic</th>
<th>Instructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>4th Sept</td>
<td>Introduction to Ensembl</td>
<td>Astrid Gall</td>
</tr>
<tr>
<td></td>
<td>Ensembl genes</td>
<td>Emily Perry</td>
</tr>
<tr>
<td>6th Sept</td>
<td>Variation data in Ensembl and the Ensembl VEP</td>
<td>Erin Haskell</td>
</tr>
<tr>
<td></td>
<td>Comparing genes and genomes with Ensembl Compara</td>
<td>Astrid Gall</td>
</tr>
<tr>
<td>11th Sept</td>
<td>Finding features that regulate genes – the Ensembl</td>
<td>Emily Perry</td>
</tr>
<tr>
<td></td>
<td>Regulatory Build</td>
<td>Erin Haskell</td>
</tr>
<tr>
<td></td>
<td>Data export with BioMart</td>
<td></td>
</tr>
<tr>
<td>13th Sept</td>
<td>Uploading your data to Ensembl</td>
<td>Astrid Gall</td>
</tr>
<tr>
<td></td>
<td>Introduction to the Ensembl REST APIs</td>
<td>Emily Perry</td>
</tr>
</tbody>
</table>
Structure

Presentation: What a REST API is and how it works

Demo: Trying out some endpoints

Take our online REST API course, if you like.
Access scales

One by one

Main browser
Mobile site

BioMart
REST API
VEP

Perl API
MySQL

Groups

FTP

Whole genome
What is a REST API?

REpresentational State Transfer. It describes how one system can communicate state with another.

Typically over HTTP(S), providing a machine readable, language agnostic method to access remote data or services.

http://rest.ensembl.org/data\textunderscore want

Gene: IRAK4, start: 43758944, end: 43789543, ...
Ensembl REST

- Language agnostic access to Ensembl datasets
- Only a fraction of the functionality of the Perl API is exposed

http://rest.ensembl.org
What Ensembl REST is and is not

- HTTP access to Ensembl data
- Stable service
- Limited by network latency
- Read only
- Versioned with archives

- Not HATEOAS†, or fully RESTful*
- No mirrors
- Not an efficient data mining solution
- Incomplete coverage

† Hypermedia As The Engine Of Application State
* See lengthy debates about Roy Fielding’s conception of REST
What is an endpoint?

“In REST, the resource typically refers to some object or set of objects that are exposed at an API endpoint. /api/users/johnny. An endpoint by itself is just a reference to a uri that accepts web requests that may or may not be RESTful. /services/service.asmx.”

An endpoint is a particular output that you can get given a particular input.
It is a function that interacts with our database.
Endpoint documentation

Full documentation of all the endpoints is found at:

http://rest.ensembl.org

The documentation lists:

• All the endpoints grouped by function
• The required parameters for each endpoint
• Optional parameters
• Example code for using the endpoints
Functional groupings

- Archive
- Comparative Genomics
- Cross References
- Information
- Lookup
- Mapping
- Ontology & Taxonomy
- Sequence
- Variation, etc...
## Endpoint Documentation

**GET lookup/id/:id**

Find the species and database for a single identifier e.g. gene, transcript, protein

### Parameters

#### Required

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
<th>Example Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>String</td>
<td>An Ensembl stable ID</td>
<td>-</td>
<td>ENSG00000157764</td>
</tr>
</tbody>
</table>

#### Optional

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
<th>Example Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>callback</td>
<td>String</td>
<td>Name of the callback subroutine to be returned by the requested JSONP response. Required ONLY when using JSONP as the serialisation method. Please see the user guide.</td>
<td>-</td>
<td>randomlygeneratedname</td>
</tr>
<tr>
<td>db_type</td>
<td>String</td>
<td>Restrict the search to a database other than the default. Useful if you need to use a DB other than core.</td>
<td>-</td>
<td>core otherfeatures</td>
</tr>
<tr>
<td>expand</td>
<td>Boolean(0,1)</td>
<td>Expands the search to include any connected features. e.g. If the object is a gene, its transcripts, translations and exons will be returned as well.</td>
<td>0</td>
<td>-</td>
</tr>
</tbody>
</table>

You must include the id in the URL in this position.

You can choose to include these in the URL in the format: `parameter=option`
Sample Code

```perl
use strict;
use warnings;

use HTTP::Tiny;

my $http = HTTP::Tiny->new();

my $server = 'http://rest.ensembl.org';
my $ext = '/lookup/id/ENSG00000157764?expand=1';

my $response = $http->get($server.$ext, {
    headers => { 'Content-type' => 'application/json' }
});

die "Failed!\n" unless $response->{success};

use JSON;
use Data::Dumper;

if(length $response->{content}) {
```
Making a REST call in the browser

• The easiest way to make REST calls is to put URLs into the browser
• This can be used as a quick look-up
• This can help you to test the URLs in your scripts to see:
  • If they work
  • If you’ve included the correct parameters
  • What the output looks like
Pinging the database

Ping confirms that you have a connection to the database

http://rest.ensembl.org/info/ping?content-type=application/json

{  
  ping: 1
}

Requesting a gene by ID

http://rest.ensembl.org/lookup/id/ENSG00000157764?content-type=application/json

```json
{
    "source": "ensembl_havana",
    "object_type": "Gene",
    "logic_name": "ensembl_havana_gene",
    "version": 12,
    "species": "homo_sapiens",
    "description": "B-Raf proto-oncogene, serine/threonine kinase [Source:HGNC Symbol;Acc:HGNC:1097]",
    "display_name": "BRAF",
    "assembly_name": "GRCh38",
    "biotype": "protein_coding",
    "end": 140924764,
    "seq_region_name": "7",
    "db_type": "core",
    "strand": -1,
    "id": "ENSG00000157764",
    "start": 140719327
}
```
Scripting around REST API calls

Scripting around calls allows you to:

• Extract specific bits of data from your REST call.
• Output in your preferred format.
• Link together calls for more complicated queries.
• Integrate your queries into a larger pipeline.
Language agnostic access

• REST APIs are designed to be accessed using any programming language.
• Calls can be made and decoded within any script.
• We have examples in Python, Perl and R.
import requests, sys, json
from pprint import pprint

def fetch_endpoint(server, request, content_type):
    r = requests.get(server+request, headers={"Content-Type": content_type})

    if not r.ok:
        r.raise_for_status()
        sys.exit()

    if content_type == 'application/json':
        return r.json()
    else:
        return r.text

server = "http://rest.ensembl.org/"
ext = "lookup/id/ENSG00000157764?"
con = "application/json"
get_gene = fetch_endpoint(server, ext, con)

pprint(get_gene)
use strict;
use warnings;

use Data::Dumper;
use HTTP::Tiny;
use JSON;

# Fetch an endpoint from the server, allow overriding of the default content type
sub fetch_endpoint {
    my $http = HTTP::Tiny->new();
    my ($server, $extension, $content_type) = @_;;
    $content_type ||= 'application/json';
    my $response = $http->get($server.$extension, { headers => { 'Accept' => $content_type } });
    die "Error: ", $response->{status}, "\n" unless $response->{success};
    if($content_type eq 'application/json') {
        return decode_json($response->{content});
    } else {
        return $response->{content};
    }
}

my $server = "http://rest.ensembl.org/";
my $ext = "lookup/id/ENSG00000157764?";
my $con = "application/json";
my $get_gene = fetch_endpoint($server, $ext, $con);

print Dumper $get_gene;
library(httr)
library(jsonlite)

fetch_endpoint <- function(server, request, content_type){
  r <- GET(paste(server, request, sep = ""), accept(content_type))
  stop_for_status(r)
  if (content_type == 'application/json'){
    return (fromJSON(content(r, "text")))
  } else {
    return (content(r, "text"))
  }
}

server <- "http://rest.ensembl.org/
ext <- "lookup/id/ENSG00000157764?"
con <- "application/json"
get_gene <- fetch_endpoint(server, ext, con)

prettify(toJSON(get_gene))
Requesting a gene by ID

http://rest.ensembl.org/lookup/id/ENSG00000157764?content-type=application/json

```json
{
    "source": "ensembl_havana",
    "object_type": "Gene",
    "logic_name": "ensembl_havana_gene",
    "version": 12,
    "species": "homo_sapiens",
    "description": "B-Raf proto-oncogene, serine/threonine kinase [Source:HGNC Symbol;Acc:HGNC:1097]",
    "display_name": "BRAF",
    "assembly_name": "GRCh38",
    "biotype": "protein_coding",
    "end": 140924764,
    "seq_region_name": "7",
    "db_type": "core",
    "strand": -1,
    "id": "ENSG00000157764",
    "start": 140719327
}
```
Decoding JSON

• Most of the time you’ll get results in JSON format
• JSON is essentially a massive dictionary/hash/dataframe with keys and values.
• Sometimes a key may then contain another nested dictionary or list
  • Which may contain another
    • And another
    • And another
• Look at the json to work out what keys you need
• You can cycle through all keys in a dictionary with for loops
Using results

Since JSON is a dictionary, you can pull out a single datapoint using the key.

```json
{
  "source": "ensembl_havana",
  "object_type": "Gene",
  "logic_name": "ensembl_havana_gene",
  "version": 12,
  "species": "homo_sapiens",
  "description": "B-Raf proto-oncogene, serine/threonine kinase [Source:HGNC Symbol;Acc:HGNC:1097]",
  "display_name": "BRAF",
  "assembly_name": "GRCh38",
  "biotype": "protein_coding",
  "end": 140924764,
  "seq_region_name": "7",
  "db_type": "core",
  "strand": -1,
  "id": "ENSG00000157764",
  "start": 140719327
}
```
Linking endpoints together

• If you can pull a datapoint from the JSON, you can use it as input for another endpoint.
• You’ll need to link objects and extensions together.
import requests, sys, json
from pprint import pprint

def fetch_endpoint(server, request, content_type):
    r = requests.get(server+request, headers={"Content-Type": content_type})

    if not r.ok:
        r.raise_for_status()
        sys.exit()

    if content_type == 'application/json':
        return r.json()
    else:
        return r.text

gene_name = "IRAK4"

server = "http://rest.ensembl.org/
ext = "lookup//symbol/homo_sapiens/" + gene_name + "?"
con = "application/json"

# submit the query
get_lookup = fetch_endpoint(server, ext_get_lookup, con)

# define the REST query to get the sequence from the gene
ext_get_seq = "/sequence/id/" + get_lookup['id'] + "?"
get_seq = fetch_endpoint(server, ext_get_seq, "text/x-fasta")

# print the gene name, ID and sequence
print (">", gene_name, "\n" + get_seq)
use strict;
use warnings;

use Data::Dumper;
use HTTP::Tiny;
use JSON;

# Fetch an endpoint from the server, allow overriding of the default content type
sub fetch_endpoint {
    my $http = HTTP::Tiny->new();
    my ($server, $extension, $content_type) = @_;
    $content_type ||= 'application/json';
    my $response = $http->get($server.$extension, { headers => { 'Accept' => $content_type } });
    die "Error: ", $response->{status}, "\n" unless $response->{success};
    if ($content_type eq 'application/json') {
        return decode_json($response->{content});
    } else {
        return $response->{content};
    }
}

my $gene_name = "IRAK4";
my $server = "http://rest.ensembl.org/";
my $ext = join("", "lookup/id/ENSG00000157764", $gene_name, "?");
my $json = "application/json";
my $get_gene = fetch_endpoint($server, $ext, $json);

# define the REST query to get the sequence from the gene
my $ext_get_seq = join("", "/sequence/id/", $get_lookup->{id}, "?");
my $fasta = "text/x-fasta";
my $get_seq = fetch_endpoint($server, $ext_get_seq, $fasta);

# print the gene name, ID and sequence
print "> ", $gene_name, "/n", $get_seq;
library(httr)
library(jsonlite)

fetch_endpoint <- function(server, request, content_type){
  r <- GET(paste(server, request, sep = ""), accept(content_type))
  stop_for_status(r)
  if (content_type == 'application/json'){
    return (fromJSON(content(r, "text")))
  } else {
    return (content(r, "text"))
  }
}

gene_name <- "IRAK4"

server <- "http://rest.ensembl.org/
con <- "application/json"
ext_get_lookup <- paste("lookup/symbol/homo_sapiens/", gene_name, "?", sep ="")

get_lookup <- fetch_endpoint(server, ext_get_lookup, con)

stable_id <- get_lookup$id

# define the REST query to get the sequence from the gene
ext_get_seq <- paste("sequence/id/", get_lookup$id, "?", sep ="")
get_seq <- fetch_endpoint(server, ext_get_seq, 'text/x-fasta')

# print the gene name, ID and sequence
paste(">", gene_name, sep = "")
get_seq
HTTP Methods - GET vs POST

GET http://rest.ensembl.org/lookup/ENSG00000157764

POST http://rest.ensembl.org/lookup/

{ "ids" : ["ENSG00000157764", "ENSG00000248378"]}
Rate limiting

Requests are rate limited to prevent a single user from monopolising the resources.

Response headers show we are allowed 55000 requests over an hour (3600 seconds)
An average 15 requests per second
1 request used and 892 sec (~15 minutes) from reset

X-RateLimit-Limit: 55000
X-RateLimit-Reset: 892
X-RateLimit-Period: 3600
X-RateLimit-Remaining: 54999
Rate limiting

Requests are rate limited to prevent a single user from monopolising the resources.

Retry-After: 40.0
X-RateLimit-Limit: 55000
X-RateLimit-Reset: 892
X-RateLimit-Period: 3600
X-RateLimit-Remaining: 54999

Wait 40 seconds before sending another request or...
Full course

• If you’re keen on working with the REST API, we have a full course online
• The course uses Jupyter notebooks to run code accessing our REST API
• You can do the course in:
  • Python
  • Perl
  • R
• https://notebooks.azure.com/ensembl-training
Questions

• We’ve muted all the mics
• Ask questions in the Slack workspace

Astrid Gall  Erin Haskell
Course exercises

http://www.ebi.ac.uk/training/online/course/ensembl-browser-webinar-series-2016

This text will be replaced by a YouTube (link to YouKu too) video of the webinar and a pdf of the slides.

A link to exercises and their solutions will appear in the page hierarchy.

The “next page” will be the exercises.
Get help with the exercises

• Use the exercise solutions in the online course
• Join our Slack workspace and discuss the exercises with everybody in dedicated channels (register to get sent a link)
• Email us helpdesk@ensembl.org
## This webinar course

<table>
<thead>
<tr>
<th>Date</th>
<th>Webinar topic</th>
<th>Instructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>4th Sept</td>
<td>Introduction to Ensembl</td>
<td>Astrid Gall</td>
</tr>
<tr>
<td></td>
<td>Ensembl genes</td>
<td>Emily Perry</td>
</tr>
<tr>
<td>6th Sept</td>
<td>Variation data in Ensembl and the Ensembl VEP</td>
<td>Erin Haskell</td>
</tr>
<tr>
<td></td>
<td>Comparing genes and genomes with Ensembl Compara</td>
<td>Astrid Gall</td>
</tr>
<tr>
<td>11th Sept</td>
<td>Finding features that regulate genes – the Ensembl Regulatory Build</td>
<td>Emily Perry</td>
</tr>
<tr>
<td></td>
<td>Data export with BioMart</td>
<td>Erin Haskell</td>
</tr>
<tr>
<td>13th Sept</td>
<td>Uploading your data to Ensembl</td>
<td>Astrid Gall</td>
</tr>
<tr>
<td></td>
<td>Introduction to the Ensembl REST APIs</td>
<td>Emily Perry</td>
</tr>
</tbody>
</table>
Help and documentation

Courses online
http://www.ebi.ac.uk/training/online/subjects/11

Tutorials
www.ensembl.org/info/website/tutorials

Flash animations
www.youtube.com/user/EnsemblHelpdesk
http://u.youku.com/Ensemblhelpdesk

Email us helpdesk@ensembl.org
Ensembl public mailing lists dev@ensembl.org, announce@ensembl.org
Publications

http://www.ensembl.org/info/about/publications.html

**Ensembl 2018**
Zerbino *et al*
http://europepmc.org/abstract/MED/29155950

Topic-specific publications mentioned throughout workshop
Ensembl 2018
Ensembl Acknowledgements

The Entire Ensembl Team

Daniel R. Zerbino\(^1\), Premanand Achuthan\(^1\), Wasiu Akanni\(^1\), M. Ridwan Amode\(^1\), Daniel Barrell\(^1,2\), Jyothish Bhai\(^1\), Konstantinos Billis\(^1\), Carla Cummins\(^1\), Astrid Gall\(^1\), Carlos García Giroñ\(^1\), Laurent Gil\(^1\), Leo Gordon\(^1\), Leanne Haggerty\(^1\), Erin Haskell\(^1\), Thibaut Hourlier\(^1\), Osagie G. Izuogu\(^1\), Sophie H. Janacek\(^1\), Thomas Juettemann\(^1\), Jimmy Kiang To\(^1\), Matthew R. Laird\(^1\), Ilia Lavidas\(^1\), Zhicheng Liu\(^1\), Jane E. Loveland\(^1\), Thomas Maurel\(^1\), William McLaren\(^1\), Benjamin Moore\(^1\), Jonathan Mudge\(^1\), Daniel N. Murphy\(^1\), Victoria Newman\(^1\), Michael Nuhn\(^1\), Denye Ogeh\(^1\), Chuang Kee Ong\(^1\), Anne Parker\(^1\), Mateus Patricio\(^1\), Harpreet Singh Riat\(^1\), Helen Schuilenburg\(^1\), Dan Sheppard\(^1\), Helen Sparrow\(^1\), Kieron Taylor\(^1\), Anja Thormann\(^1\), Alessandro Vullo\(^1\), Brandon Walts\(^1\), Amonida Zadissa\(^1\), Adam Frankish\(^1\), Sarah E. Hunt\(^1\), Myrto Kostadima\(^1\), Nicholas Langridge\(^1\), Fergal J. Martin\(^1\), Matthieu Muffato\(^1\), Emily Perry\(^1\), Magali Ruffier\(^1\), Dan M. Staines\(^1\), Stephen J. Trevanion\(^1\), Bronwen L. Aken\(^1\), Fiona Cunningham\(^1\), Andrew Yates\(^1\) and Paul Flicek\(^1,3\)

\(^1\)European Molecular Biology Laboratory, European Bioinformatics Institute, Wellcome Genome Campus, Hinxton, Cambridge CB10 1SD, UK, \(^2\)Eagle Genomics Ltd., Wellcome Genome Campus, Hinxton, Cambridge CB10 1DR, UK and \(^3\)Wellcome Trust Sanger Institute, Wellcome Genome Campus, Hinxton, Cambridge CB10 1SA, UK

Funding