


Training materials

- Ensembl training materials are protected by a CC BY license 
- <http://creativecommons.org/licenses/by/4.0/>
- If you wish to re-use these materials, please credit Ensembl for their creation
- If you use Ensembl for your work, please cite our papers
- <http://www.ensembl.org/info/about/publications.html>

Ensembl REST API course

Ben Moore

EMBL-EBI



e!Ensembl

Course agenda

- Ensembl and the gene model
- What is REST
- Ensembl REST server features
- Fetching a single endpoint
- Decoding the response to link together endpoints
- POST endpoints
- Rate limiting

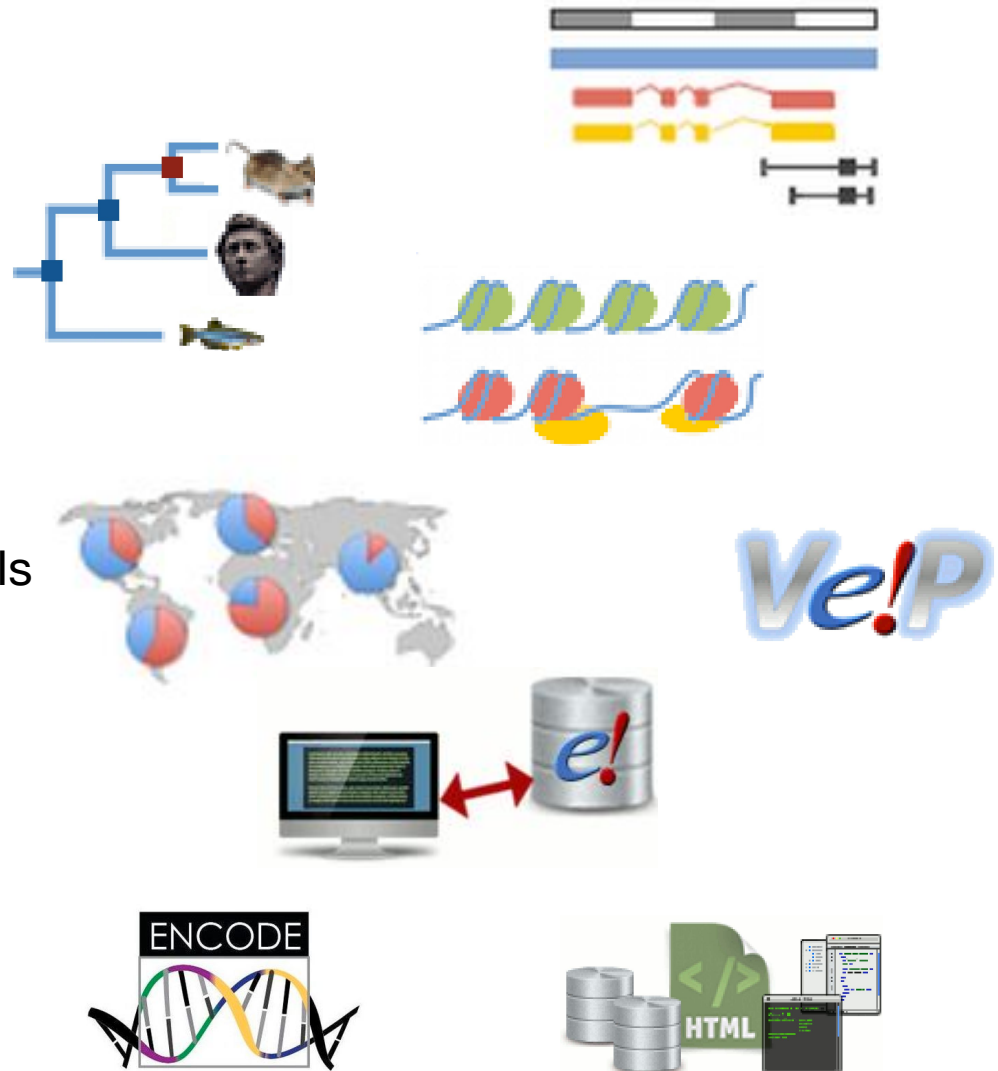
Course materials

<http://training.ensembl.org/events/>

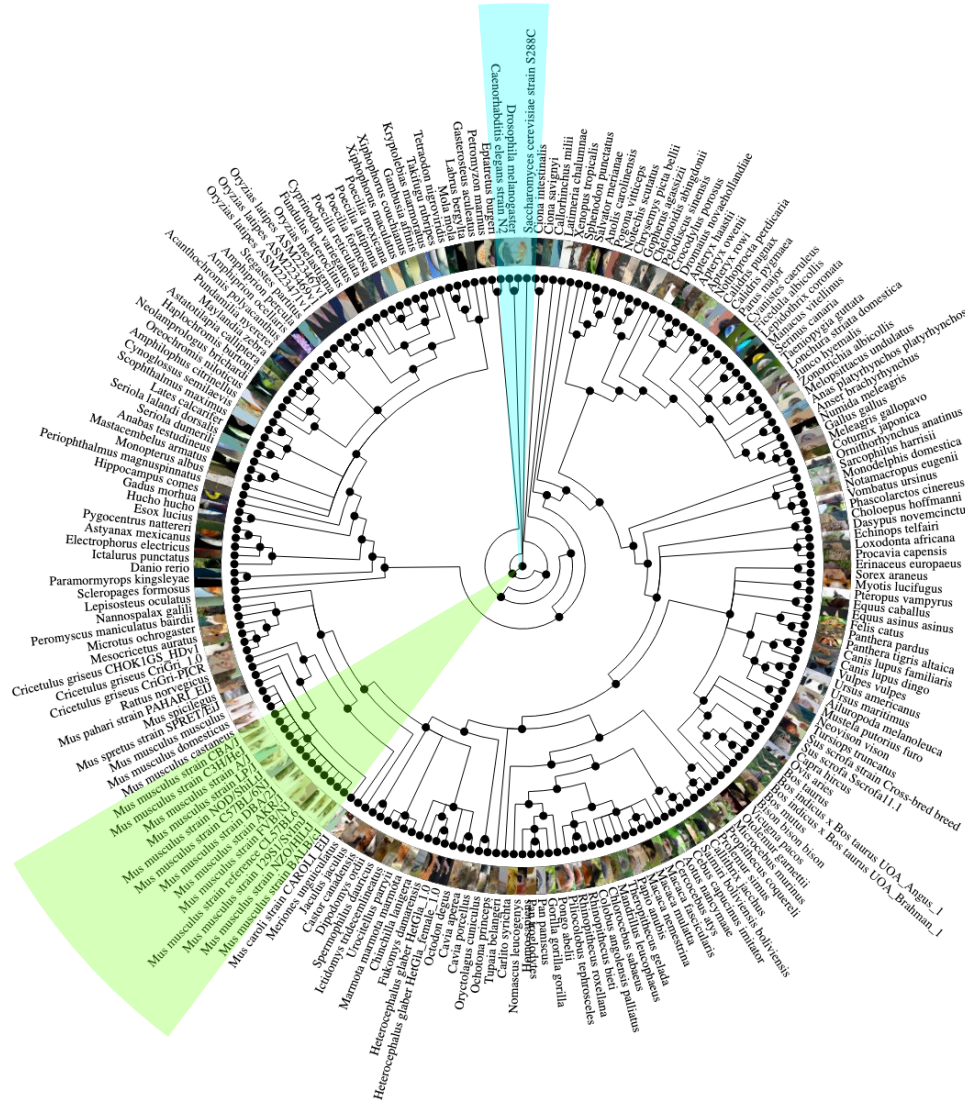
- Slides
- Notebooks in Python and R
 - Use whichever notebook you feel comfortable with
 - You will need to clone it with your Microsoft Account
- When we demo the example answers we will use Python only

Ensembl Features

- Gene builds for >200 species
- Gene trees
- Regulatory build (ENCODE)
- Variation display and VEP
- Display of user data
- BioMart (data export)
- Programmatic access via the APIs
- Completely Open Source



Vertebrate species on Ensembl



www.ensembl.org



Non-vertebrates on Ensembl genomes

Ensembl Bacteria

Find a Species

Ensembl Bacteria Species

Bacillus collection
78 genomes

Bacillus amyloliquefaciens European Nucleotide Archive	Bacillus anthracis A0248 European Nucleotide Archive	Bacillus anthracis Ames European Nucleotide Archive
Bacillus anthracis Ames ancestor European Nucleotide Archive	Bacillus anthracis CDC 684 European Nucleotide Archive	Bacillus anthracis Sterne European Nucleotide Archive
Bacillus cereus 0308102 European Nucleotide Archive	Bacillus cereus 172660W European Nucleotide Archive	Bacillus cereus 95/6201 European Nucleotide Archive
Bacillus cereus AH1271 European Nucleotide Archive	Bacillus cereus AH1272 European Nucleotide Archive	Bacillus cereus AH1273 European Nucleotide Archive
Bacillus cereus AH187 European Nucleotide Archive	Bacillus cereus AH603 European Nucleotide Archive	Bacillus cereus AH621 European Nucleotide Archive
Bacillus cereus AH676 European Nucleotide Archive	Bacillus cereus AH820 European Nucleotide Archive	Bacillus cereus ATCC 10876 European Nucleotide Archive
Bacillus cereus ATCC 10987 European Nucleotide Archive	Bacillus cereus ATCC 14579 European Nucleotide Archive	Bacillus cereus ATCC 4342 European Nucleotide Archive
Bacillus cereus B4264 European Nucleotide Archive	Bacillus cereus BDRD-Bc4 European Nucleotide Archive	Bacillus cereus BDRD-ST196 European Nucleotide Archive
Bacillus cereus BDRD-ST24 European Nucleotide Archive	Bacillus cereus BDRD-ST26 European Nucleotide Archive	Bacillus cereus BGSC 6E1 European Nucleotide Archive
Bacillus cereus F65185 European Nucleotide Archive	Bacillus cereus G0842 European Nucleotide Archive	Bacillus cereus MM3 European Nucleotide Archive

Bacteria

Ensembl Protists

Find a Species

Ensembl Protists Species

Alveolates

Plasmodium berghei GeneDB Plasmodium berghei ANKA	Plasmodium knowlesi Wellcome Trust Sanger Institute Plasmodium knowlesi	Toxoplasma gondii ToxoDB Toxoplasma gon
Plasmodium chabaudi GeneDB Plasmodium chabaudi	Plasmodium vivax The Institute for Genomic Research Plasmodium vivax	
Plasmodium falciparum GeneDB Plasmodium falciparum 3D7	Tetrahymena thermophila The Institute for Genomic Research Tetrahymena thermophila SB210	

Amoebozoa

Dictyostellum discoideum DictyBase Dictyostellum discoideum	Entamoeba histolytica AmoeboDB Entamoeba histolytica HM-1:IMSS	
---	--	--

Stramenopiles

Albugo laibachii The Sainsbury Laboratory Albugo laibachii Nc14	Phytophthora infestans BROAD Phytophthora infestans	Pythium ultimum Pythium Genome Database Pythium ultimum
---	---	---

Protists

Ensembl Fungi

Find a Species

Ensembl Fungi Species

Capnodiales

Mycosphaerella graminicola JGI Mycosphaerella graminicola IPO323		
---	--	--

Eurotiales

Aspergillus clavatus CADRE Aspergillus clavatus	Aspergillus fumigatus CADRE Aspergillus fumigatus A1163	Aspergillus oryzae CADRE Aspergillus oryzae
Aspergillus flavus CADRE Aspergillus flavus	Aspergillus nidulans FGSC 44	Aspergillus terreus CADRE Aspergillus terreus
Aspergillus fumigatus CADRE Aspergillus fumigatus A1293	Aspergillus niger CBS 513.88	Neosartorya fischeri CADRE Neosartorya fischeri

Hypocerales

Fusarium oxysporum Broad Institute Fusarium oxysporum 4287	Gibberella zeae zeae PH-1	Trichoderma virens JGI Trichoderma virens (C-8)
--	------------------------------	--

Fungi

Ensembl Metazoa

Find a Species

Ensembl Metazoa Species

Diptera

Aedes aegypti VectorBase Aedes aegypti	Drosophila grimshawi FlyBase Drosophila grimshawi	Drosophila simulans FlyBase Drosophila simulans
Anopheles darlingi European Nucleotide Archive Anopheles darlingi	Drosophila melanogaster FlyBase Drosophila melanogaster	Drosophila virilis FlyBase Drosophila virilis
Anopheles gambiae VectorBase Anopheles gambiae	Drosophila mojavensis FlyBase Drosophila mojavensis	Drosophila willistoni FlyBase Drosophila willistoni
Culex quinquefasciatus VectorBase Culex quinquefasciatus	Drosophila persimilis FlyBase Drosophila persimilis	Drosophila yakuba FlyBase Drosophila yakuba
Drosophila ananassae FlyBase Drosophila ananassae	Drosophila pseudoobscura FlyBase Drosophila pseudoobscura	
Drosophila erecta FlyBase Drosophila erecta	Drosophila sechellia FlyBase Drosophila sechellia	

Metazoa

Ensembl Plants

Find a Species

Ensembl Plants Species

Liliopsida

Brachypodium distachyon Brachypodium.org Brachypodium distachyon (L.) Beauv	Oryza glaberrima AGI Oryza glaberrima	Sorghum bicolor JGI Sorghum bicolor BTx
Hordeum vulgare BSG Hordeum vulgare	Oryza sativa MSU Oryza sativa Nipponbare (Japonica rice)	Zea mays MaizeSequence Zea mays
Musa acuminata Oryza Musa acuminata Doubled-haploid Pahaung (DH- Pahaung)	Oryza sativa Indica Group IRIS Oryza sativa 92-11 (Indica rice)	
Oryza brachyantha OGE Oryza brachyantha	Setaria italica JGI Setaria italica	

euclitoyledons

Arabidopsis lyrata JGI Arabidopsis lyrata	Glycine max JGI Glycine max	Solanum tuberosum PGSC Solanum tuberosum
Arabidopsis thaliana TAIR Arabidopsis thaliana	Populus trichocarpa JGI Populus trichocarpa	Vitis vinifera Genoscope Vitis vinifera

Plants



www.ensemblgenomes.org



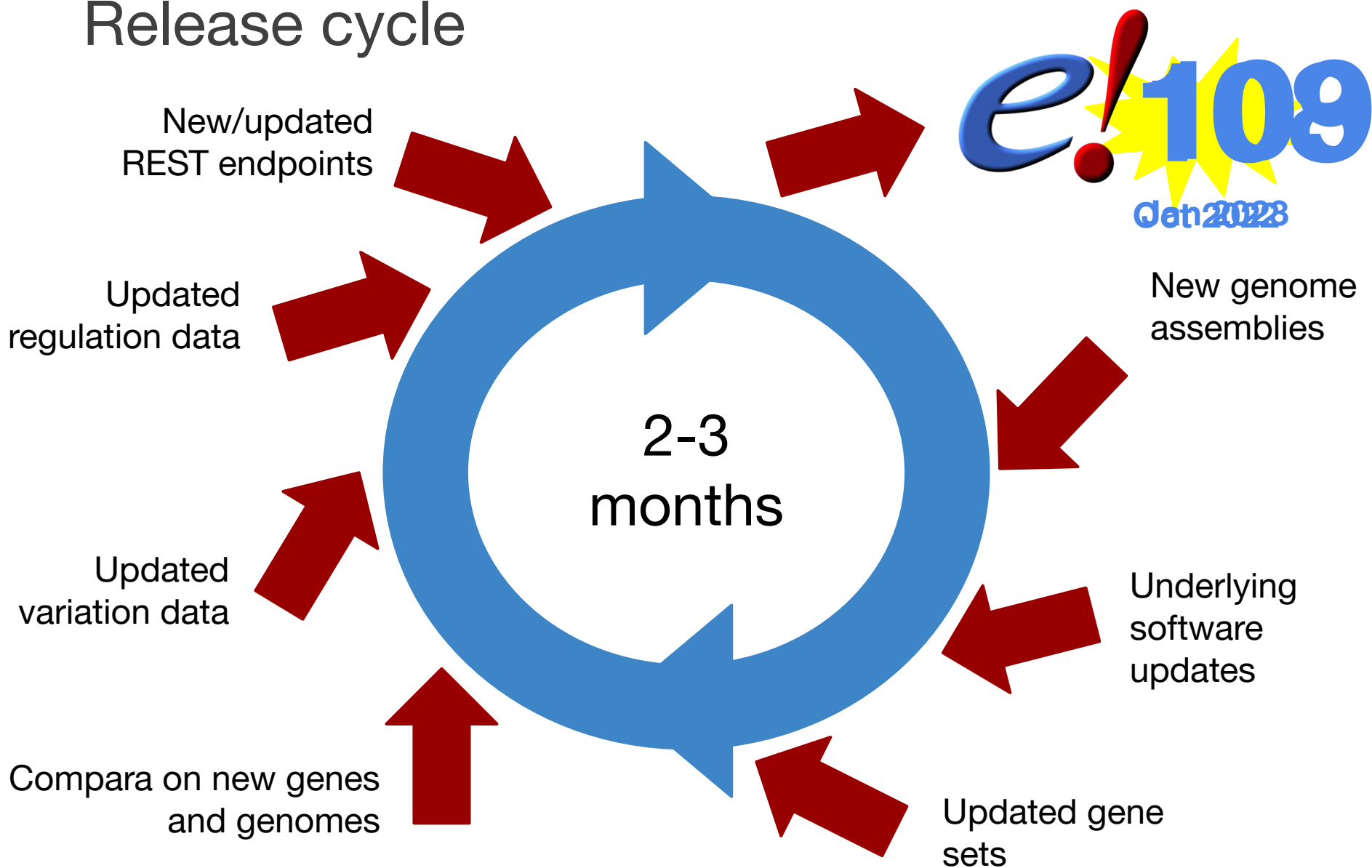
Ensembl and Ensembl Genomes

	Ensembl	EnsemblGenomes
Released	2000	2009
Species	Vertebrates (fly, worm and yeast as outgroups)	Non-vertebrates (protists, plants, fungi, metazoa, bacteria)
Annotation	by Ensembl	in collaboration with the scientific communities
URL	rest.ensembl.org	rest.ensembl.org

Human genome assemblies

- GRCh38 (aka hg38) 
 - No gaps. Many rare/private alleles replaced.
 - rest.ensembl.org
 - Software regularly updated
 - Data regularly updated
- GRCh37 (aka hg19) 
 - 250 gaps
 - grch37.rest.ensembl.org
 - Software regularly updated
 - Data only rarely updated

Release cycle

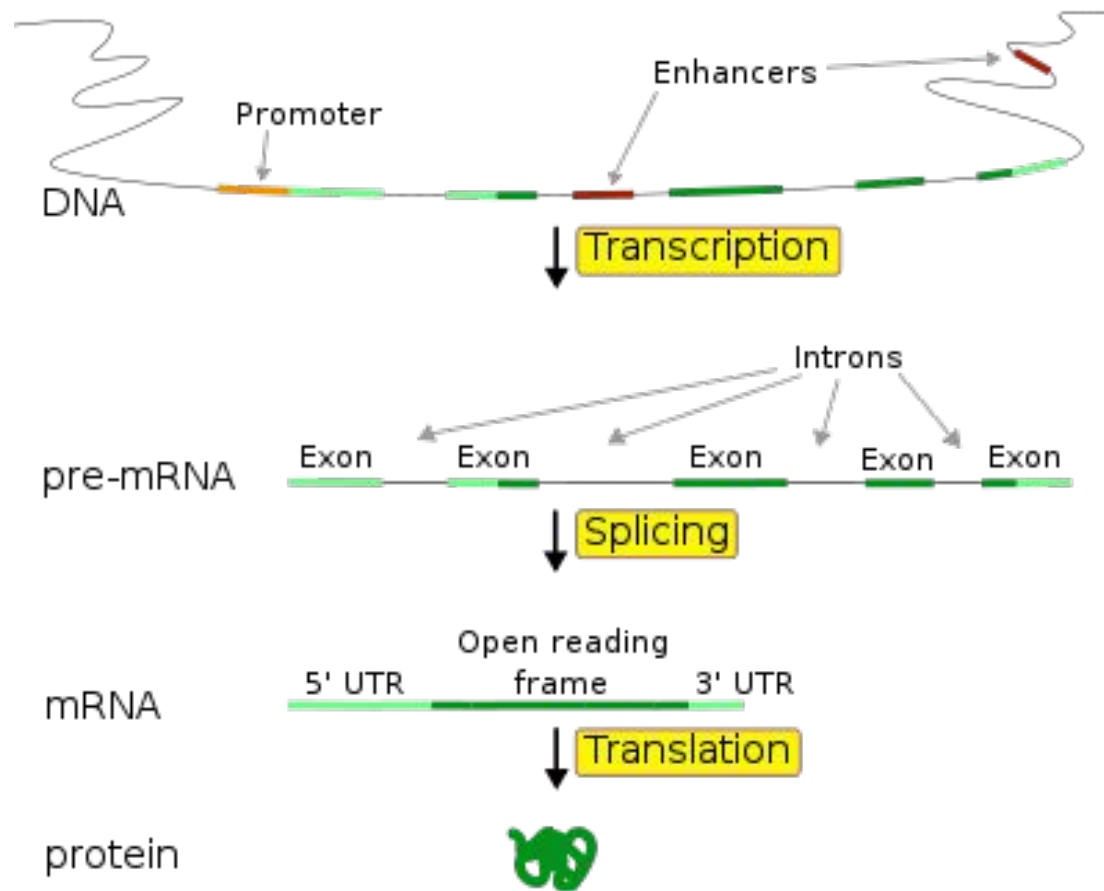


REST Archives

Starting with release 87, there are REST archives (GRCh38 only). We will continue to provide archive services for up to five years, to match the Ensembl website archives.

<http://e87.rest.ensembl.org>

Ensembl Data Model

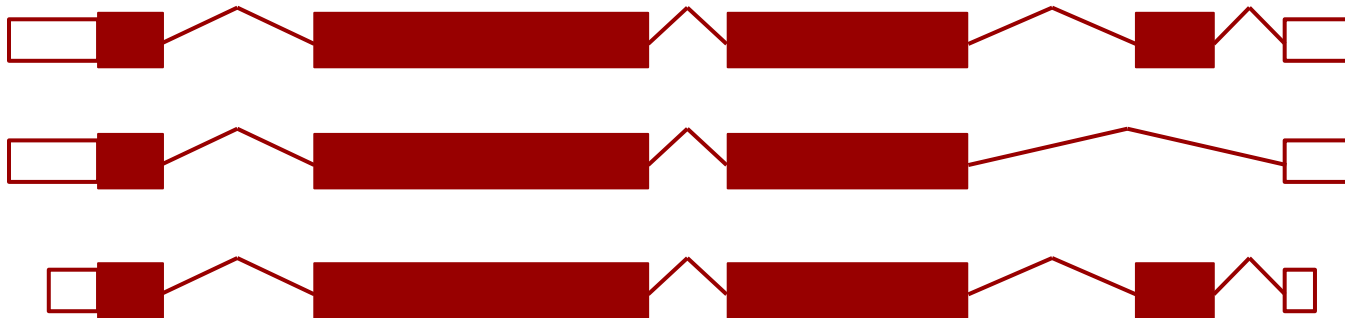


Ensembl Data Model

Primary feature types of Genes, Transcript, and Exons

A Gene is a set of alternatively spliced Transcripts

A Transcript is a set of Exons



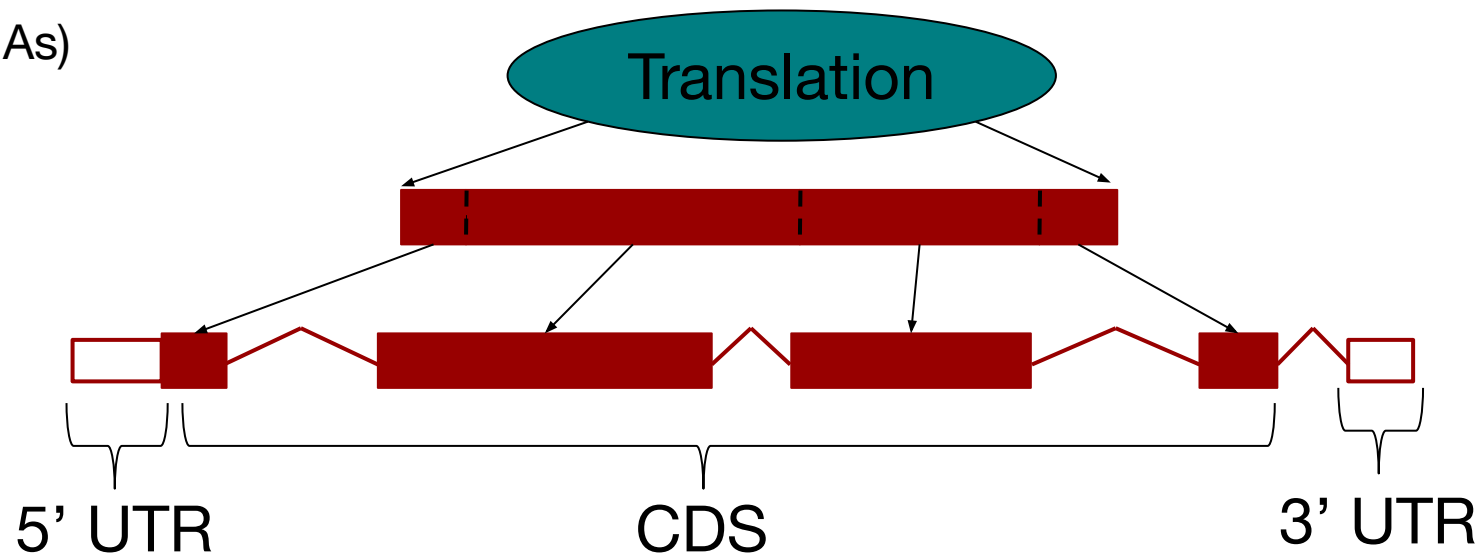
Ensembl Data Model

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 Transcript objects.

Not all transcripts have a translation
(e.g. ncRNAs)

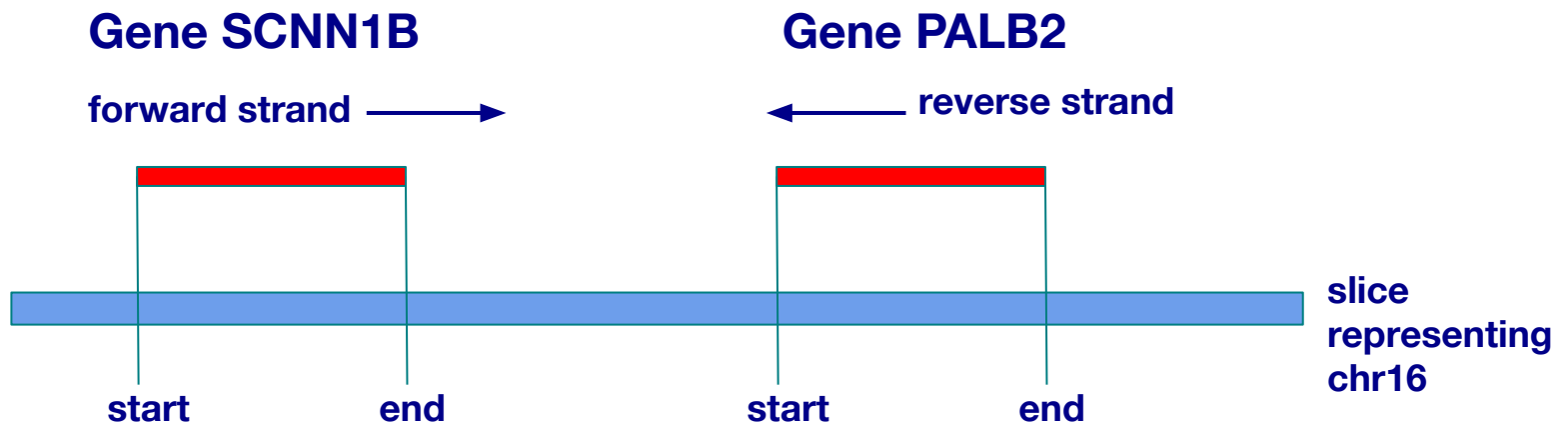


Features

Features have a defined location on the genome

Start and end are always plotted on the forward strand

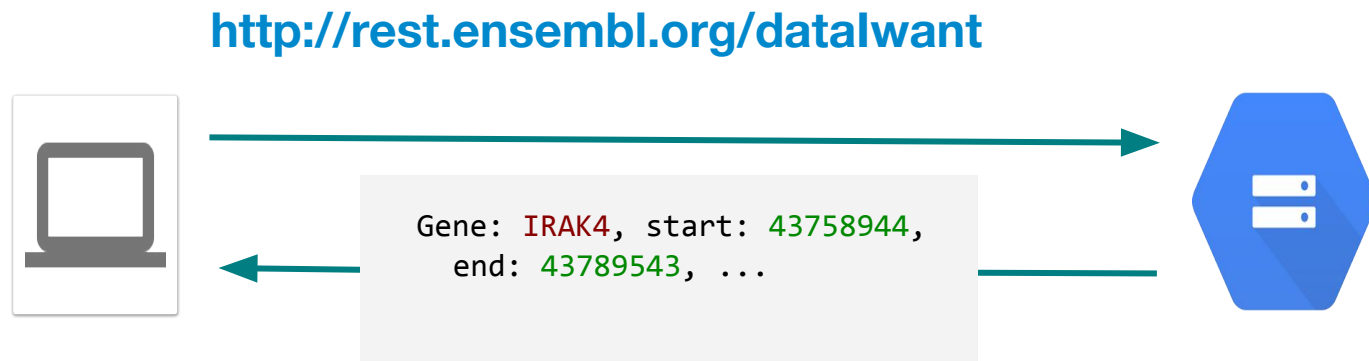
start < end



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.



Ensembl REST

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

<http://rest.ensembl.org>

Ensembl REST API Endpoints	
Archive	
Resource	Description
GET archive/id/id	Uses the given identifier to return the archived sequence
POST archive/id	Retrieve the archived sequence for a set of identifiers
Comparative Genomics	
Resource	Description
GET geneteel/id/id	Retrieves a gene tree for a gene tree stable identifier
GET geneteel/member/id/id	Retrieves the gene tree that contains the gene / transcript / translation stable identifier
GET geneteel/member/symbol/species/symbol	Retrieves the gene tree that contains the gene identified by a symbol
GET alignment/region/species/region	Retrieves genomic alignments as separate blocks based on a region and species
GET homology/id/id	Retrieves homology information (orthologs) by Ensembl gene id
GET homology/symbol/species/symbol	Retrieves homology information (orthologs) by symbol
Cross References	
Resource	Description
GET xrefs/symbol/species/symbol	Looks up an external symbol and returns all Ensembl objects linked to it. This can be a display name for a gene/transcript/translation, a synonym or an externally linked reference. If a gene's transcript is linked to the supplied symbol the service will return both gene and transcript (it supports transient links).
GET xrefs/id/id	Perform lookups of Ensembl identifiers and retrieve their external references in other databases
GET xrefs/name/species/name	Performs a lookup based upon the primary accession or display label of an external reference and returning the information

What Ensembl REST is and is not

- + HTTP access to Ensembl data
- + Stable service
- + Limited by network latency
- + Read only
- + Versioned with archives
- No mirrors
- Not an efficient data mining solution
- Incomplete coverage

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:

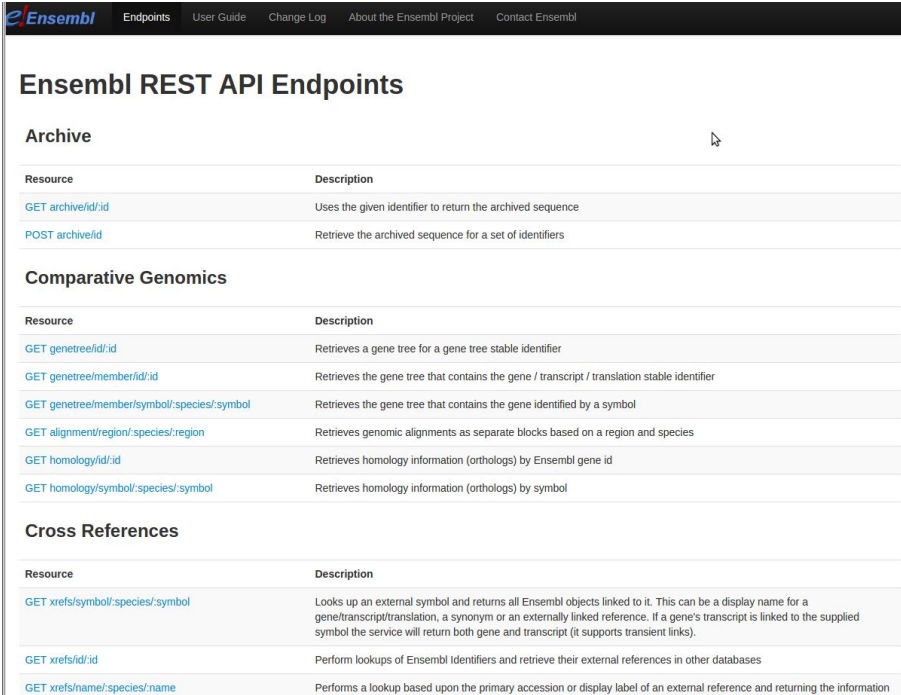
<http://rest.ensembl.org>

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...



The screenshot shows the Ensembl REST API Endpoints page. It features a navigation bar at the top with links to 'Endpoints', 'User Guide', 'Change Log', 'About the Ensembl Project', and 'Contact Ensembl'. The main content is organized into three sections: 'Archive', 'Comparative Genomics', and 'Cross References'. Each section contains a table with two columns: 'Resource' and 'Description'.

Ensembl REST API Endpoints	
Archive	
Resource	Description
GET archive/id/id	Uses the given identifier to return the archived sequence
POST archive/id	Retrieve the archived sequence for a set of identifiers
Comparative Genomics	
Resource	Description
GET genetree/id/id	Retrieves a gene tree for a gene tree stable identifier
GET genetree/member/id/id	Retrieves the gene tree that contains the gene / transcript / translation stable identifier
GET genetree/member/symbol/species/symbol	Retrieves the gene tree that contains the gene identified by a symbol
GET alignment/region/species/region	Retrieves genomic alignments as separate blocks based on a region and species
GET homology/id/id	Retrieves homology information (orthologs) by Ensembl gene id
GET homology/symbol/species/symbol	Retrieves homology information (orthologs) by symbol
Cross References	
Resource	Description
GET xrefs/symbol/species/symbol	Looks up an external symbol and returns all Ensembl objects linked to it. This can be a display name for a gene/transcript/translation, a synonym or an externally linked reference. If a gene's transcript is linked to the supplied symbol the service will return both gene and transcript (it supports transient links).
GET xrefs/id/id	Perform lookups of Ensembl Identifiers and retrieve their external references in other databases
GET xrefs/name/species/name	Performs a lookup based upon the primary accession or display label of an external reference and returning the information

Endpoint Documentation

GET lookup/id/:id

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

You must include the id in the URL in this position

Parameters

Required

Name	Type	Description	Default	Example Values
id	String	An Ensembl stable ID	-	ENSG00000157764

Optional

Name	Type	Description	Default	Example Values
callback	String	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 .	-	randomlygeneratedname
db_type	String	Restrict the search to a database other than the default. Useful if you need to use a DB other than core	-	core otherfeatures
expand	Boolean(0,1)	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.	0	-

Resource Information

Methods	GET
Response formats	json xml jsonp

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

Sample Code

Example output

Perl

Python2

Python3

Ruby

Java

Curl

Wget

```
1. use strict;
2. use warnings;
3.
4. use HTTP::Tiny;
5.
6. my $http = HTTP::Tiny->new();
7.
8. my $server = 'http://rest.ensembl.org';
9. my $ext = '/lookup/id/ENSG00000157764?expand=1';
10. my $response = $http->get($server.$ext, {
11.     headers => { 'Content-type' => 'application/json' }
12. });
13.
14. die "Failed!\n" unless $response->{success};
15.
16.
17. use JSON;
18. use Data::Dumper;
19. 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  
}
```

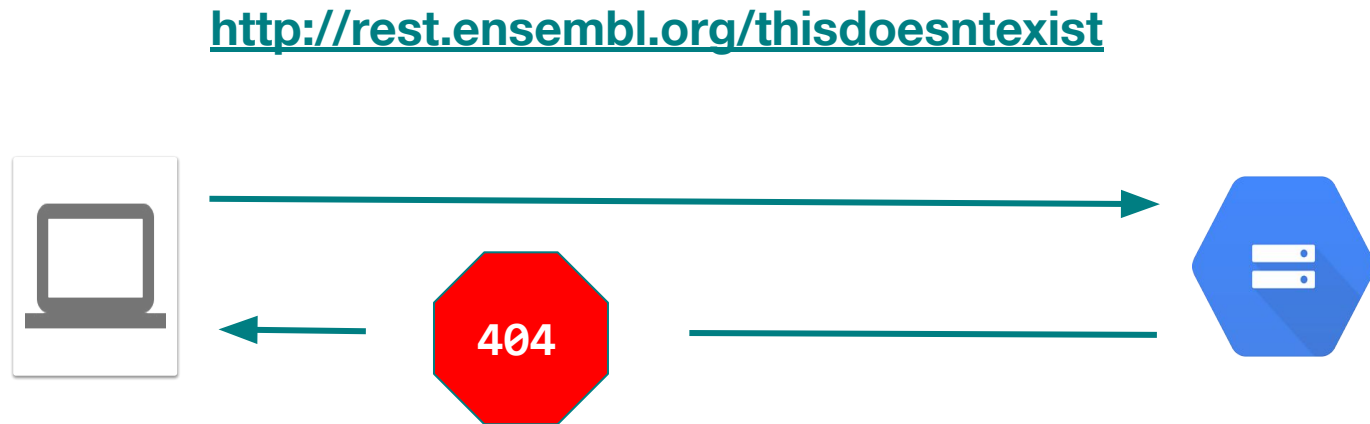
Requesting a gene by ID

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

```
{
  "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
}
```


HTTP Status Codes

The server uses HTTP status codes to signal the request outcome



HTTP Status Codes

Code	Name	Notes
200	OK	Request was a success
400	Bad Request	Occurs during exceptional circumstances such as the service is unable to find an ID. Check if the response Content-type or Accept was JSON. If so the JSON object is an exception hash with the message keyed under error
403	Forbidden	You are submitting far too many requests and have been temporarily forbidden access to the service. Wait and retry with a maximum of 15 requests per second.
404	Not Found	Indicates a badly formatted request. Check your URL

<https://github.com/Ensembl/ensembl-rest/wiki/HTTP-Response-Codes>

HTTP Status Codes (cont.)

Code	Name	Notes
408	Timeout	The request was not processed in time. Wait and retry later
429	Too Many Requests	You have been rate-limited; wait and retry. The headers <code>X-RateLimit-Reset</code> , <code>X-RateLimit-Limit</code> and <code>X-RateLimit-Remaining</code> will inform you of how long you have until your limit is reset and what that limit was. If you get this response and have not exceeded your limit then check if you have made too many requests per second.
503	Service Unavailable	The service is temporarily down; retry after a pause
418	I'm a teapot	An April Fools joke added in 1998, who said computer scientists don't have a sense of humour?

<https://github.com/Ensembl/ensembl-rest/wiki/HTTP-Response-Codes>

Exercises 1

1. Find an endpoint which you can use to **lookup** information about a gene using its symbol.
2. Create a URL to find information about the gene *ESPN* in human.
3. **Expand** your results to include information about transcripts.

Answers 1

1. http://rest.ensembl.org/documentation/info/symbol_lookup
2. http://rest.ensembl.org/lookup/symbol/homo_sapiens/ESPN?content-type=application/json
3. http://rest.ensembl.org/lookup/symbol/homo_sapiens/ESPN?content-type=application/json;expand=1

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.

Python modules

- To make requests in Python, you will need the **requests** package:
 - <http://docs.python-requests.org/en/master/user/install/> (not needed for this course, this is all set up in your Python Notebook)
- To decode JSON you will need the **JSON** package:
 - Should ship with standard Python installations
- You'll need **pprint** to print JSON in an easy to read way
 - Should ship with standard Python installations

```
import requests, sys, json
from pprint import pprint
```

R libraries

- To make requests in R you will need the [httr](#) library
- To decode JSON you'll need the [jsonlite](#) package

```
library(httr)  
library(jsonlite)
```

Requesting a gene by ID

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

Making a request – Python

- Make a string of the server (you'll use this multiple times)
- Make another string of the extension with all the parameters

```
import requests, sys

server = "http://rest.ensembl.org"
ext = "/lookup/id/ENSG00000157764?expand=1"

r = requests.get(server+ext, headers={ "Accept" : "application/json"})

pprint (r)
```

Making a request – R

- Make a string of the server (you'll use this multiple times)
- Make another string of the extension with all the parameters

```
library(httr)
library(jsonlite)

server <- "http://rest.ensembl.org"
ext <- "/lookup/id/ENSG00000157764"

r <- GET(paste(server, ext, sep = ""), accept("application/json"))

r
```

Error handling – Python

You should never assume a response will return correctly.

```
import requests, sys

server = "http://rest.ensembl.org"
ext = "/lookup/id/ENSG00000157764?expand=1"

r = requests.get(server+ext, headers={ "Accept" : "application/json"})

if not r.ok:
    r.raise_for_status()
```

Check the response code returned by the server.

Error handling – R

You should never assume a response will return correctly.

```
library(httr)
library(jsonlite)

server <- "http://rest.ensembl.org"
ext <- "/lookup/id/ENSG00000157764"

r <- GET(paste(server, ext, sep = ""), content_type("application/json"))

r

stop_for_status(r)
```

Check the response code returned by the server.

Accept

HTTP allows the serving of different representations of a resource based on client preferences

Content-type and Accept headers are how servers and clients negotiate what format they will communicate with.

`text/html`, `text/plain`, `application/json`, `image/png`, etc.

Accept

Resource Information

Methods	GET
Response formats	fasta json seqxml text yaml jsonp

- The returned content types can be specified in the header as accept (you'll need to use content-type in URLs)
- Endpoint documentation pages list allowed content-types

<https://github.com/Ensembl/ensembl-rest/wiki/Output-formats>

Decoding the response – Python

- In most cases you'll be using JSON formatted responses
- Most languages have JSON parsers that return the data as a structure
- In Python pretty print (`pprint`) will give you a human readable format

```
decoded = r.json()
```

```
pprint (decoded)
```

Decoding the response – R

- In most cases you'll be using JSON formatted responses
- Most languages have JSON parsers that return the data as a structure
- In R `pretty` will give you a human readable format

```
decoded = content(r, "text")
```

```
pretty (decoded)
```

Decoding JSON

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

- 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

Helper function

- The helper function in your python script makes your life easier by:
 - Calling the request with the specified server, extension and content type.
 - Getting the status of a failed query
 - Decoding the JSON (if you've used JSON as your content type)
 - Returning the text (if you use any other content type)
- Add it to every script then just call it when you need to fetch an endpoint

Python Helper function

```
def fetch_endpoint(server, request, content_type):  
    """  
    Fetch an endpoint from the server, allow overriding of default content-type  
    """  
    r = requests.get(server+request, headers={ "Accept" : content_type})  
  
    if not r.ok:  
        r.raise_for_status()  
        sys.exit()  
  
    if content_type == 'application/json':  
        return r.json()  
    else:  
        return r.text
```


R Helper function

```
Fetch_endpoint <- function(server, request, content_type){  
  ""  
  Fetch an endpoint from the server, allow overriding of default 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"))  
  }  
}
```

Exercises 2

1. Write a script to **lookup** the gene called *ESPN* in human and print the results in JSON.

Using results

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

```
{
  "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
}
```

Using results – Python

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

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

symbol = get_gene['display_name']
print (symbol)
```

Using results – R

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

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

symbol <- get_gene$display_name
symbol
```

Nested JSON lists

<http://rest.ensembl.org/overlap/region/human/7:140424943-140444564?feature=gene;content-type=application/json>

```
[  
  {  
    "gene_id": "ENSG00000146955"  
    "Feature_type": "gene",  
    "external_name": "RAB19",  
    "description": "RAB19, member RAS oncogene family [Source:HGNC  
    Symbol;Acc:HGNC:19982]",  
    "Biotype": "protein_coding",  
    "id": "ENSG00000146955",  
  },  
  {  
    "gene_id": "ENSG00000103200",  
    "Feature_type": "gene",  
    "external_name": "AC069335.1",  
    "Description": null,  
    "Biotype": "processed_pseudogene"  
    "id": "ENSG00000103200"  
  }  
]
```

List delineated by square brackets [] – no keys

Dictionary delineated by curly brackets {} – key-value pairs

Exercises 3

1. Write a script to **lookup** the gene called *ESPN* in human and print the stable **ID** of this gene.
2. Get all variants that are associated with the phenotype 'Coffee consumption'. For each variant print
 - a. the p-value for the association
 - b. the PMID for the publication which describes the association between that variant and 'Coffee consumption'
 - c. the risk allele and the associated gene.
3. Get the mouse **homologue** of the human *BRCA2* and print the ID and **sequence** of both.

Other content types – Python

- If you specify another content type (not JSON), the helper function will get you this as text
- This can be used to get:
 - Sequence in FASTA
 - Gene trees and homologues in various formats
 - Alignments

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


Other content types – R

- If you specify another content type (not JSON), the helper function will get you this as text
- This can be used to get:
 - Sequence in FASTA
 - Gene trees and homologues in various formats
 - Alignments

```
if (content_type == 'application/json'){  
  return (fromJSON(content(r, "text")))  
} else {  
  return (content(r, "text"))  
}
```

Other content types

Resource Information

Methods	GET
Response formats	fasta json seqxml text yaml jsonp

- Endpoint documentation pages list allowed content-types
- The wiki lists how you specify these

<https://github.com/Ensembl/ensembl-rest/wiki/Output-formats>

Exercises 4

1. Get the **gene tree** predicted for the gene ENSG00000189221 in full nh format.
2. Get the **sequence** of the gene ENSG00000157764 in FASTA.

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.

Exercises 5

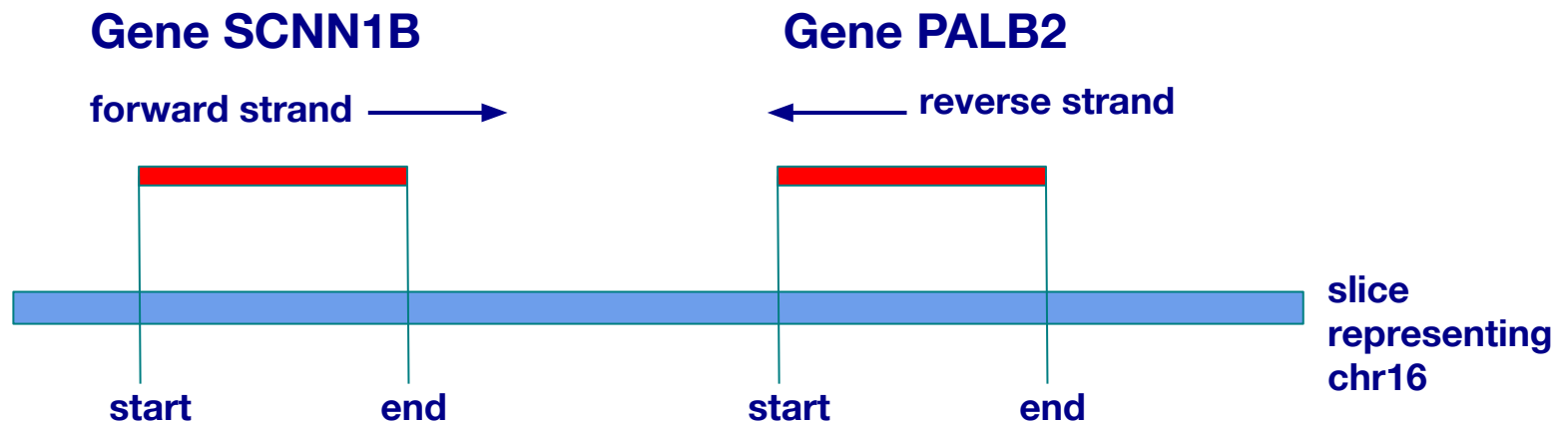
1. Using the script from 3.1, add a call to fetch and print the **sequence** for the gene *ESPN* in FASTA.
2. Print the stable ID of any regulatory features that **overlap** the region 1000 bp upstream of the *ESPN* gene. (Hints: get the gene ID first, then check the strand of the gene to see which way is upstream.)

Features

Features have a defined location on the genome

Start and end are always plotted on the forward strand

start < end

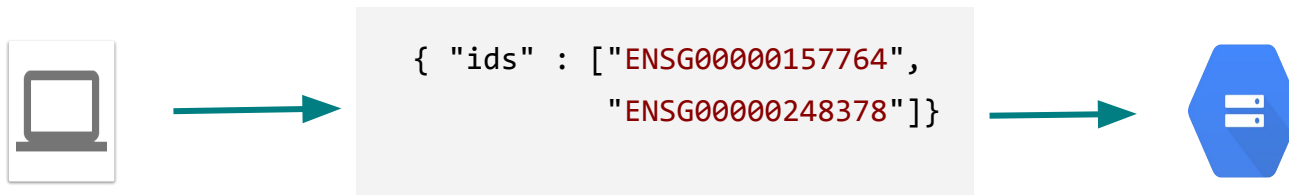


HTTP Methods - GET vs POST

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



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



Using POST - Python

```
import requests, sys

server = "http://rest.ensembl.org"
ext = "/lookup/id"
headers={ "Content-Type" : "application/json", "Accept" : "application/json"}
r = requests.post(server+ext, headers=headers, data='{ "ids" :
["ENSG00000157764", "ENSG00000248378" ] }')

# error checking removed for space

decoded = r.json()
pprint (decoded)
```


Using POST - R

```
library(httr)
library(jsonlite)

server <- "http://rest.ensembl.org"
ext <- "/lookup/id"
genes <- c("ENSG00000157764", "ENSG00000248378")
body_values <- toJSON(list(ids=genes))

r <- POST(paste(server, ext, sep = ""), content_type("application/json"),
accept("application/json"), body = body_values)

prettify(content(r, "text"))
```

Helper function

- You can also have a helper function for POST queries
- You'll need to create a list of your values
- If you have a Python list you can convert it to a JSON list with:

```
data = json.dumps({ "ids" : my_list })
```

- R

```
data <- toJSON(list(ids=mylist))
```

Python Helper function

```
def fetch_endpoint_POST(server, request, data, content_type='application/json'):

    r = requests.post(server+request,
                      headers={ "Content-Type" : content_type},
                      data=data )

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

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

R Helper function

```
fetch_endpoint_POST <- function(server, request, content_type){  
  ""  
  Fetch an endpoint from the server, allow overriding of default content-type  
  ""  
  r <- POST(paste(server, request, sep = ""), content_type(content_type),  
    accept(content_type), body = data)  
  
  stop_for_status(r)  
  
  if (content_type == 'application/json'){  
    return (fromJSON(content(r, "text")))  
  } else {  
    return (content(r, "text"))  
  }  
}
```

Decoding POST queries

POST endpoints return a dictionary of dictionaries.

```
{
  "ENSG00000157764": {
    "source": "ensembl_havana",
    "object_type": "Gene",
    ...
  },
  "ENSG00000248378": {
    "source": "havana",
    "object_type": "Gene",
    ...
  }
}
```

Decoding POST queries

- You could use your input list as your keys, or you could move through the dictionary with:

- Python

```
for key, value in post_query.items():
```

- Perl

```
foreach my $hash_reference  
(@{$post_query}) {  
    my %hash = %$hash_reference;  
}
```

- R just treats these as dataframes

Exercises 6

1. Fetch the all the transcripts of *ESPN* using the **lookup** function. Fetch the cDNA **sequences** of all transcripts using a single **POST** request, and print in FASTA format.
2. Get all variants that are located on chromosome 17 between 80348215 and 80348333. Get the variant class, evidence attributes, source and the `most_severe_consequence` for all variants in that region from the variant POST endpoint.

Rate limiting

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

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

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

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...

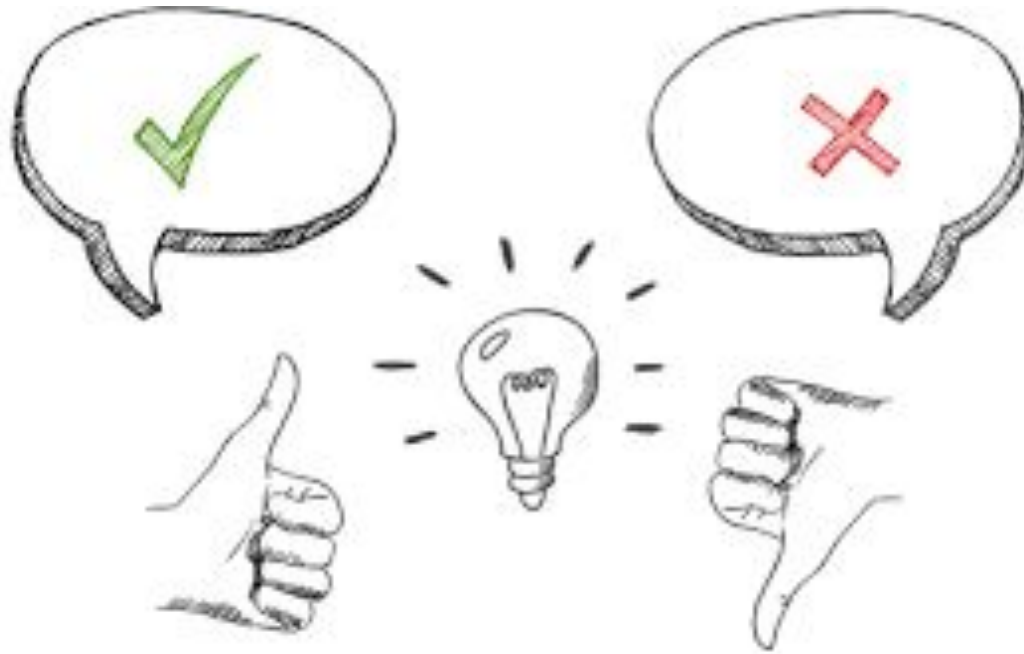
429

Exercises 7

The Jupyter notebook contains a script that queries the *ping* endpoint 25 times, printing the count, the HTTP Status Code, and the X-RateLimit-Remaining header each time.

1. Increase the number of loops, do you start to get 429 errors?
2. Can you add in a step to make it wait a few seconds every iteration? Or every 100 iterations?

Feedback



training.ensembl.org/events

Help and Documentation



The REST API release notes:

<https://github.com/Ensembl/ensembl-rest/wiki/Change-log>



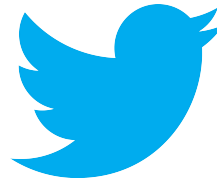
Email us helpdesk@ensembl.org

Ensembl public mailing lists dev@ensembl.org,
announce@ensembl.org

Follow us



www.facebook.com/Ensembl.org



[@Ensembl](https://twitter.com/Ensembl)
[@BENsembl](https://twitter.com/BENsembl)



www.ensembl.info



Ensembl Acknowledgements

The Entire Ensembl Team

Adam Frankish, Ahamed Imran Abdul Salam, Alexandra Bignell, Ameya Chaubal, Andrea Winterbottom, Andrew Berry, Andrew Parton, Andrey Azov, Andy Yates, Anja Thormann, Anmol Jaywant Hemrom, Anne Lyle, Astrid Gall, Benjamin Moore, Bethany Flint, Brandon Walts, Bruno Contreras-Moreira, Carla Cummins, Carlos Garcia Giron, Claire Davidson, Cristina Guijarro, Dan Sheppard, Daniel Zerbino, David Thybert, Denye Ogeh, Diana Lemos, Elizabeth Lewis, Emily Perry, Fergal Martin, Fiona Cunningham, Gareth Maslen, Gareth Williams, Garth Ilesley, Guy Naamati, Helen Schuilenburg, IF Barnes, Ilias Lavidas, Irina Armean, James Allen, Jamie Allen, Jane Loveland, Jonathan Mudge, Jorge Alvarez-Jarreta, Jose Carlos Marugan, Jose Manuel Gonzalez Martinez, Jyothish Bhai, Kamalkumar Jayantilal Dodiya, Kevin Howe, Kieron Taylor, Kostas Billis, Lahcen Campbell, Leanne Haggerty, Luca Da Rin Fioretto, Magali Ruffier, Manoj Sakthivel, Manuel Carbajo Martinez, Marc Chakiachvili, Marek Szuba, Marie-Marthe Suner, Matthew Hardy, Matthew Russell, Matthieu Barba, Matthieu Muffato, Michael Paulini, Michal Szpak, Mike Kay, Mikkel Christensen, Mira Sycheva, Nick Langridge, Nishadi De Silva, Osagie Izuogu, Paul Davis, Paul Flicek, Premanand Achuthan, Reham Fatima, Ridwan Amode, Ruth Bennett, Sanjay Boddu, Sarah Donaldson, Sarah Hunt, Shamika Mohanan, Stephen Trevanion, Thibaut Hourlier, Thomas Juettemann, Thomas Maurel, Tiago Grego, Toby Hunt, Tuan Le, Vasili Sitnik

Funding



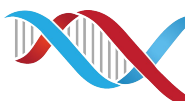
EMBL



National
Human Genome
Research Institute



Open Targets




TRANSFORMING GENETIC
MEDICINE INITIATIVE



Co-funded by the
European Union

Training materials

- Ensembl training materials are protected by a CC BY license 
- <http://creativecommons.org/licenses/by/4.0/>
- If you wish to re-use these materials, please credit Ensembl for their creation
- If you use Ensembl for your work, please cite our papers
- <http://www.ensembl.org/info/about/publications.html>