pyensembl Documentation

Release 0.8.10

Hammer Lab

Oct 16, 2019

Contents

1 pyensembl **3**
1.1 pyensembl package 3

2 Indices and tables **27**

Python Module Index **29**

Index **31**

Contents:

CHAPTER 1

pyensembl

1.1 pyensembl package

1.1.1 Submodules

1.1.2 pyensembl.biotypes module

1.1.3 pyensembl.common module

pyensembl.common.**dump_pickle**(*obj*, *filepath*)

pyensembl.common.**load_pickle**(*filepath*)

pyensembl.common.**memoize**(*fn*)

Simple reset-able memoization decorator for functions and methods, assumes that all arguments to the function can be hashed and compared.

1.1.4 pyensembl.database module

class pyensembl.database.**Database**(*gtf_path*, *install_string=None*, *cache_directory_path=None*, *restrict_gtf_columns=None*, *restrict_gtf_features=None*)

Bases: object

Wrapper around sqlite3 database so that the rest of the library doesn't have to worry about constructing the .db file or writing SQL queries directly.

PRIMARY_KEY_COLUMNS = {'gene': 'gene_id', 'transcript': 'transcript_id'}

column_exists(*table_name*, *column_name*)

column_values_at_locus(*column_name*, *feature*, *contig*, *position*, *end=None*, *strand=None*, *distinct=False*, *sorted=False*)

Get the non-null values of a column from the database at a particular range of loci

columns(*table_name*)

connect_or_create(*overwrite=False*)
Return a connection to the database if it exists, otherwise create it. Overwrite the existing database if *overwrite* is True.

connection
Get a connection to the database or raise an exception

create(*overwrite=False*)
Create the local database (including indexing) if it's not already set up. If *overwrite* is True, always re-create the database from scratch.

Returns a connection to the database.

distinct_column_values_at_locus(*column*, *feature*, *contig*, *position*, *end=None*, *strand=None*)
Gather all the distinct values for a property/column at some specified locus.

column [str] Which property are we getting the values of.

feature [str] Which type of entry (e.g. transcript, exon, gene) is the property associated with?

contig [str] Chromosome or unplaced contig name

position [int] Chromosomal position

end [int, optional] End position of a range, if unspecified assume we're only looking at the single given position.

strand [str, optional] Either the positive ('+') or negative strand ('-'). If unspecified then check for values on either strand.

local_db_filename

local_db_path

query(***kwargs*)
Construct a SQL query and run against the sqlite3 database, filtered both by the feature type and a user-provided column/value.

query_distinct_on_contig(*column_name*, *feature*, *contig*)

query_feature_values(***kwargs*)
Run a SQL query against the sqlite3 database, filtered only on the feature type.

query_loci(*filter_column*, *filter_value*, *feature*)
Query for loci satisfying a given filter and feature type.

filter_column [str] Name of column to filter results by.

filter_value [str] Only return loci which have this value in the their filter_column.

feature [str] Feature names such as 'transcript', 'gene', and 'exon'

Returns list of Locus objects

query_locus(*filter_column*, *filter_value*, *feature*)
Query for unique locus, raises error if missing or more than one locus in the database.

filter_column [str] Name of column to filter results by.

filter_value [str] Only return loci which have this value in the their filter_column.

feature [str] Feature names such as 'transcript', 'gene', and 'exon'

Returns single Locus object.

query_one(*select_column_names*, *filter_column*, *filter_value*, *feature*, *distinct=False*, *required=False*)

run_sql_query(*sql*, *required=False*, *query_params=[]*)

Given an arbitrary SQL query, run it against the database and return the results.

sql [str] SQL query

required [bool] Raise an error if no results found in the database

query_params [list] For each '?' in the query there must be a corresponding value in this list.

1.1.5 pyensembl.download_cache module

class pyensembl.download_cache.**DownloadCache**(*reference_name*, *annotation_name*, *annotation_version=None*, *decompress_on_download=False*, *copy_local_files_to_cache=False*, *install_string_function=None*, *cache_directory_path=None*)

Bases: object

Downloads remote files to cache, optionally copies local files into cache, raises custom message if data is missing.

cache_directory_path

cached_path(*path_or_url*)

When downloading remote files, the default behavior is to name local files the same as their remote counterparts.

delete_cache_directory()

delete_cached_files(*prefixes=[]*, *suffixes=[]*)

Deletes any cached files matching the prefixes or suffixes given

download_or_copy_if_necessary(*path_or_url*, *download_if_missing=False*, *overwrite=False*)

Download a remote file or copy Get the local path to a possibly remote file.

Download if file is missing from the cache directory and *download_if_missing* is True. Download even if local file exists if both *download_if_missing* and *overwrite* are True.

If the file is on the local file system then return its path, unless self.copy_local_to_cache is True, and then copy it to the cache first.

path_or_url : str

download_if_missing [bool, optional] Download files if missing from local cache

overwrite [bool, optional] Overwrite existing copy if it exists

is_url_format(*path_or_url*)

Is the given string a URL?

path_or_url : str

bool

local_path_or_install_error(*field_name*, *path_or_url*, *download_if_missing=False*, *overwrite=False*)

exception pyensembl.download_cache.**MissingLocalFile**(*path*)

Bases: exceptions.Exception

exception pyensembl.download_cache.**MissingRemoteFile**(*url*)
Bases: exceptions.Exception

pyensembl.download_cache.**cache_subdirectory**(*reference_name=None*, *annotation_name=None*, *annotation_version=None*)
Which cache subdirectory to use for a given annotation database over a particular reference. All arguments can be omitted to just get the base subdirectory for all pyensembl cached datasets.

1.1.6 pyensembl.ensembl_release module

Contains the EnsemblRelease class, which extends the Genome class to be specific to (a particular release of) Ensembl.

class pyensembl.ensembl_release.**EnsemblRelease**(*release=97*, *species=Species(latin_name='homo_sapiens', synonyms=['human'], reference_assemblies={'GRCh38': (76, 97), 'GRCh37': (55, 75), 'NCBI36': (54, 54)}), server='ftp://ftp.ensembl.org'*)
Bases: *pyensembl.genome.Genome*

Bundles together the genomic annotation and sequence data associated with a particular release of the Ensembl database.

classmethod cached(*release=97*, *species=Species(latin_name='homo_sapiens', synonyms=['human'], reference_assemblies={'GRCh38': (76, 97), 'GRCh37': (55, 75), 'NCBI36': (54, 54)}), server='ftp://ftp.ensembl.org'*)
Construct EnsemblRelease if it's never been made before, otherwise return an old instance.

classmethod from_dict(*state_dict*)
Deserialize EnsemblRelease without creating duplicate instances.

install_string()
Add every missing file to the install string shown to the user in an error message.

classmethod normalize_init_values(*release*, *species*, *server*)
Normalizes the arguments which uniquely specify an EnsemblRelease genome.

to_dict()
Returns a dictionary of the essential fields of this Genome.

pyensembl.ensembl_release.**cached_release**(*release*, *species='human'*)
Create an EnsemblRelease instance only if it's hasn't already been made, otherwise returns the old instance. Keeping this function for backwards compatibility but this functionality has been moving into the cached method of EnsemblRelease.

1.1.7 pyensembl.ensembl_release_versions module

pyensembl.ensembl_release_versions.**check_release_number**(*release*)
Check to make sure a release is in the valid range of Ensembl releases.

1.1.8 pyensembl.ensembl_url_templates module

Templates for URLs and paths to specific relase, species, and file type on the Ensembl ftp server.

For example, the human chromosomal DNA sequences for release 78 are in:

ftp://ftp.ensembl.org/pub/release-78/fasta/homo_sapiens/dna/

pyensembl.ensembl_url_templates.**make_fasta_filename**(*ensembl_release*, *species*, *sequence_type*)

pyensembl.ensembl_url_templates.**make_fasta_url**(*ensembl_release*, *species*, *sequence_type*, *server='ftp://ftp.ensembl.org'*)

Construct URL to FASTA file with cDNA transcript or protein sequences

Parameter examples: ensembl_release = 75 species = "Homo_sapiens" sequence_type = "cdna" (other option: "pep")

pyensembl.ensembl_url_templates.**make_gtf_filename**(*ensembl_release*, *species*)

Return GTF filename expect on Ensembl FTP server for a specific species/release combination

pyensembl.ensembl_url_templates.**make_gtf_url**(*ensembl_release*, *species*, *server='ftp://ftp.ensembl.org'*)

Returns a URL and a filename, which can be joined together.

pyensembl.ensembl_url_templates.**normalize_release_properties**(*ensembl_release*, *species*)

Make sure a given release is valid, normalize it to be an integer, normalize the species name, and get its associated reference.

1.1.9 pyensembl.exon module

class pyensembl.exon.**Exon**(*exon_id*, *contig*, *start*, *end*, *strand*, *gene_name*, *gene_id*)

Bases: *pyensembl.locus.Locus*

id

Alias for exon_id necessary for backward compatibility.

to_dict()

Returns a dictionary which can be used to reconstruct an instance of a derived class (typically by matching args to __init__). The values of the returned dictionary must be primitive atomic types (bool, string, int, float), primitive collections (int, list, tuple, set) or instances of Serializable.

The default implementation is to assume all the arguments to __init__ have fields of the same name on a serializable object.

1.1.10 pyensembl.gene module

class pyensembl.gene.**Gene**(*gene_id*, *gene_name*, *contig*, *start*, *end*, *strand*, *biotype*, *genome*)

Bases: pyensembl.locus_with_genome.LocusWithGenome

exons

id

Alias for gene_id necessary for backwards compatibility.

name

Alias for gene_name necessary for backwards compatibility.

to_dict()

Returns a dictionary which can be used to reconstruct an instance of a derived class (typically by matching args to __init__). The values of the returned dictionary must be primitive atomic types (bool, string, int, float), primitive collections (int, list, tuple, set) or instances of Serializable.

The default implementation is to assume all the arguments to __init__ have fields of the same name on a serializable object.

transcripts
Property which dynamically construct transcript objects for all transcript IDs associated with this gene.

1.1.11 pyensembl.genome module

Contains the Genome class, with its millions of accessors and wrappers around an arbitrary genomic database.

class pyensembl.genome.**Genome**(*reference_name*, *annotation_name*, *annotation_version=None*, *gtf_path_or_url=None*, *transcript_fasta_paths_or_urls=None*, *protein_fasta_paths_or_urls=None*, *decompress_on_download=False*, *copy_local_files_to_cache=False*, *cache_directory_path=None*)
Bases: serializable.serializable.Serializable

Bundles together the genomic annotation and sequence data associated with a particular genomic database source (e.g. a single Ensembl release) and provides a wide variety of helper methods for accessing this data.

clear_cache()
Clear any in-memory cached values and short-lived on-disk materializations from MemoryCache

contigs()
Returns all contig names for any gene in the genome (field called "seqname" in Ensembl GTF files)

db

delete_index_files()
Delete all data aside from source GTF and FASTA files

download(*overwrite=False*)
Download data files needed by this Genome instance.

overwrite [bool, optional] Download files regardless whether local copy already exists.

exon_by_id(*exon_id*)
Construct an Exon object from its ID by looking up the exon"s properties in the given Database.

exon_ids(*contig=None*, *strand=None*)

exon_ids_at_locus(*contig*, *position*, *end=None*, *strand=None*)

exon_ids_of_gene_id(*gene_id*)

exon_ids_of_gene_name(*gene_name*)

exon_ids_of_transcript_id(*transcript_id*)

exon_ids_of_transcript_name(*transcript_name*)

exons(*contig=None*, *strand=None*)
Create exon object for all exons in the database, optionally restrict to a particular chromosome using the *contig* argument.

exons_at_locus(*contig*, *position*, *end=None*, *strand=None*)

gene_by_id(*gene_id*)
Construct a Gene object for the given gene ID.

gene_by_protein_id(*protein_id*)
Get the gene ID associated with the given protein ID, return its Gene object

gene_id_of_protein_id(*protein_id*)
What is the gene ID associated with a given protein ID?

gene_ids(*contig=None*, *strand=None*)
What are all the gene IDs (optionally restrict to a given chromosome/contig and/or strand)

gene_ids_at_locus(*contig*, *position*, *end=None*, *strand=None*)

gene_ids_of_gene_name(*gene_name*)
What are the gene IDs associated with a given gene name? (due to copy events, there might be multiple genes per name)

gene_name_of_exon_id(*exon_id*)

gene_name_of_gene_id(*gene_id*)

gene_name_of_transcript_id(*transcript_id*)

gene_name_of_transcript_name(*transcript_name*)

gene_names(*contig=None*, *strand=None*)
Return all genes in the database, optionally restrict to a chromosome and/or strand.

gene_names_at_locus(*contig*, *position*, *end=None*, *strand=None*)

genes(*contig=None*, *strand=None*)
Returns all Gene objects in the database. Can be restricted to a particular contig/chromosome and strand by the following arguments:

contig [str] Only return genes on the given contig.

strand [str] Only return genes on this strand.

genes_at_locus(*contig*, *position*, *end=None*, *strand=None*)

genes_by_name(*gene_name*)
Get all the unqiue genes with the given name (there might be multiple due to copies in the genome), return a list containing a Gene object for each distinct ID.

index(*overwrite=False*)
Assuming that all necessary data for this Genome has been downloaded, generate the GTF database and save efficient representation of FASTA sequence files.

install_string()
Add every missing file to the install string shown to the user in an error message.

loci_of_gene_names(*gene_name*)

Given a gene name returns list of Locus objects with fields: chromosome, start, stop, strand

You can get multiple results since a gene might have multiple copies in the genome.

locus_of_exon_id(*exon_id*)
Given an exon ID returns Locus

locus_of_gene_id(*gene_id*)
Given a gene ID returns Locus with: chromosome, start, stop, strand

locus_of_transcript_id(*transcript_id*)

protein_ids(*contig=None*, *strand=None*)
What are all the protein IDs (optionally restrict to a given chromosome and/or strand)

protein_ids_at_locus(*contig*, *position*, *end=None*, *strand=None*)

protein_sequence(*protein_id*)
Return cDNA nucleotide sequence of transcript, or None if transcript doesn't have cDNA sequence.

protein_sequences

required_local_files()

required_local_files_exist(*empty_files_ok=False*)

requires_gtf

requires_protein_fasta

requires_transcript_fasta

to_dict()
Returns a dictionary of the essential fields of this Genome.

transcript_by_id(*transcript_id*)
Construct Transcript object with given transcript ID

transcript_by_protein_id(*protein_id*)

transcript_id_of_protein_id(*protein_id*)
What is the transcript ID associated with a given protein ID?

transcript_ids(*contig=None*, *strand=None*)

transcript_ids_at_locus(*contig*, *position*, *end=None*, *strand=None*)

transcript_ids_of_exon_id(*exon_id*)

transcript_ids_of_gene_id(*gene_id*)

transcript_ids_of_gene_name(*gene_name*)

transcript_ids_of_transcript_name(*transcript_name*)

transcript_name_of_transcript_id(*transcript_id*)

transcript_names(*contig=None*, *strand=None*)
What are all the transcript names in the database (optionally, restrict to a given chromosome and/or strand)

transcript_names_at_locus(*contig*, *position*, *end=None*, *strand=None*)

transcript_names_of_gene_name(*gene_name*)

transcript_sequence(*transcript_id*)
Return cDNA nucleotide sequence of transcript, or None if transcript doesn't have cDNA sequence.

transcript_sequences

transcripts(*contig=None*, *strand=None*)
Construct Transcript object for every transcript entry in the database. Optionally restrict to a particular chromosome using the *contig* argument.

transcripts_at_locus(*contig*, *position*, *end=None*, *strand=None*)

transcripts_by_name(*transcript_name*)

1.1.12 pyensembl.gtf module

1.1.13 pyensembl.locus module

class pyensembl.locus.**Locus**(*contig*, *start*, *end*, *strand*)
Bases: serializable.serializable.Serializable

Base class for any entity which can be localized at a range of positions on a particular strand of a chromosome/contig.

can_overlap(*contig*, *strand=None*)
Is this locus on the same contig and (optionally) on the same strand?

contains(*contig*, *start*, *end*, *strand=None*)

contains_locus(*other_locus*)

distance_to_interval(*start*, *end*)
Find the distance between intervals [start1, end1] and [start2, end2]. If the intervals overlap then the distance is 0.

distance_to_locus(*other*)

length

offset(*position*)
Offset of given position from stranded start of this locus.

For example, if a Locus goes from 10..20 and is on the negative strand, then the offset of position 13 is 7, whereas if the Locus is on the positive strand, then the offset is 3.

offset_range(*start*, *end*)
Database start/end entries are always ordered such that start < end. This makes computing a relative position (e.g. of a stop codon relative to its transcript) complicated since the "end" position of a backwards locus is actually earlir on the strand. This function correctly selects a start vs. end value depending on this locuses's strand and determines that position's offset from the earliest position in this locus.

on_backward_strand

on_contig(*contig*)

on_forward_strand

on_negative_strand

on_positive_strand

on_strand(*strand*)

overlaps(*contig*, *start*, *end*, *strand=None*)
Does this locus overlap with a given range of positions?

Since locus position ranges are inclusive, we should make sure that e.g. chr1:10-10 overlaps with chr1:10-10

overlaps_locus(*other_locus*)

to_dict()
Returns a dictionary which can be used to reconstruct an instance of a derived class (typically by matching args to __init__). The values of the returned dictionary must be primitive atomic types (bool, string, int, float), primitive collections (int, list, tuple, set) or instances of Serializable.

The default implementation is to assume all the arguments to __init__ have fields of the same name on a serializable object.

to_tuple()

1.1.14 pyensembl.memory_cache module

Cache and serializing the results of expensive computations. Used in pyensembl primarily to cache the heavy-weight parsing of GTF files and various filtering operations on Ensembl entries.

A piece of data is returned from one of three sources: 1) Cache cold. Run the user-supplied compute_fn. 2) Cache warm on disk. Parse or unpickle the serialized result into memory. 3) Cache warm in memory. Return cached object.

class pyensembl.memory_cache.**MemoryCache**
Bases: object

In-memory and on-disk caching of long-running queries and computations.

cached_dataframe(*csv_path*, *compute_fn*)
If a CSV path is in the _memory_cache, then return that cached value.

If we've already saved the DataFrame as a CSV then load it.

Otherwise run the provided *compute_fn*, and store its result in memory and and save it as a CSV.

cached_object(*path*, *compute_fn*)
If *cached_object* has already been called for a value of *path* in this running Python instance, then it should have a cached value in the

_memory_cache; return that value.

If this function was never called before with a particular value of *path*, then call compute_fn, and pickle it to *path*.

If *path* already exists, unpickle it and store that value in _memory_cache.

clear_cached_objects()

delete_file(*path*)

is_empty(*filename*)

remove_from_cache(*key*)

1.1.15 pyensembl.search module

pyensembl.search.**find_nearest_locus**(*start*, *end*, *loci*)
Finds nearest locus (object with method *distance_to_interval*) to the interval defined by the given *start* and *end* positions. Returns the distance to that locus, along with the locus object itself.

1.1.16 pyensembl.sequence_data module

class pyensembl.sequence_data.**SequenceData**(*fasta_paths*, *cache_directory_path=None*)
Bases: object

Container for reference nucleotide and amino acid sequenes.

clear_cache()

fasta_dictionary

get(*sequence_id*)
Get sequence associated with given ID or return None if missing

index(*overwrite=False*)

1.1.17 pyensembl.shell module

Manipulate pyensembl's local cache.

%(prog)s {install, delete, delete-sequence-cache} [–release XXX –species human...]

To install particular Ensembl human release(s): %(prog)s install –release 75 77

To install particular Ensembl mouse release(s): %(prog)s install –release 75 77 –species mouse

To delete all downloaded and cached data for a particular Ensembl release: %(prog)s delete-all-files –release 75 –species human

To delete only cached data related to transcript and protein sequences: %(prog)s delete-index-files –release 75

To list all installed genomes: %(prog)s list

To install a genome from source files: %(prog)s install –reference-name "GRCh38" –gtf URL_OR_PATH –transcript-fasta URL_OR_PATH –protein-fasta URL_OR_PATH

pyensembl.shell.**all_combinations_of_ensembl_genomes**(*args*)
Use all combinations of species and release versions specified by the commandline arguments to return a list of EnsemblRelease or Genome objects. The results will typically be of type EnsemblRelease unless the –custom-mirror argument was given.

pyensembl.shell.**collect_all_installed_ensembl_releases**()

pyensembl.shell.**collect_selected_genomes**(*args*)

pyensembl.shell.**run**()

1.1.18 pyensembl.species module

class pyensembl.species.**Species**(*latin_name*, *synonyms=[]*, *reference_assemblies={}*)
Bases: serializable.serializable.Serializable

Container for combined information about a species name, its synonyn names and which reference to use for this species in each Ensembl release.

classmethod all_registered_latin_names()
Returns latin name of every registered species.

classmethod all_species_release_pairs()
Generator which yields (species, release) pairs for all possible combinations.

classmethod from_dict(*state_dict*)
Given a dictionary of flattened fields (result of calling to_dict()), returns an instance.

classmethod register(*latin_name*, *synonyms*, *reference_assemblies*)
Create a Species object from the given arguments and enter into all the dicts used to look the species up by its fields.

to_dict()
Returns a dictionary which can be used to reconstruct an instance of a derived class (typically by matching args to __init__). The values of the returned dictionary must be primitive atomic types (bool, string, int, float), primitive collections (int, list, tuple, set) or instances of Serializable.

The default implementation is to assume all the arguments to __init__ have fields of the same name on a serializable object.

which_reference(*ensembl_release*)

pyensembl.species.**check_species_object**(*species_name_or_object*)
Helper for validating user supplied species names or objects.

pyensembl.species.**find_species_by_name**(*species_name*)

pyensembl.species.**normalize_species_name**(*name*)
If species name was "Homo sapiens" then replace spaces with underscores and return "homo_sapiens". Also replace common names like "human" with "homo_sapiens".

1.1.19 pyensembl.transcript module

class pyensembl.transcript.**Transcript**(*transcript_id*, *transcript_name*, *contig*, *start*, *end*, *strand*, *biotype*, *gene_id*, *genome*, *support_level=None*)
Bases: pyensembl.locus_with_genome.LocusWithGenome

Transcript encompasses the locus, exons, and sequence of a transcript.

Lazily fetches sequence in case we"re constructing many Transcripts and not using the sequence, avoid the memory/performance overhead of fetching and storing sequences from a FASTA file.

coding_sequence
cDNA coding sequence (from start codon to stop codon, without any introns)

coding_sequence_position_ranges
Return absolute chromosome position ranges for CDS fragments of this transcript

complete
Consider a transcript complete if it has start and stop codons and a coding sequence whose length is divisible by 3

contains_start_codon
Does this transcript have an annotated start_codon entry?

contains_stop_codon
Does this transcript have an annotated stop_codon entry?

exon_intervals
List of (start,end) tuples for each exon of this transcript, in the order specified by the 'exon_number' column of the exon table.

exons

first_start_codon_spliced_offset
Offset of first nucleotide in start codon into the spliced mRNA (excluding introns)

five_prime_utr_sequence
cDNA sequence of 5' UTR (untranslated region at the beginning of the transcript)

gene

gene_name

id
Alias for transcript_id necessary for backward compatibility.

last_stop_codon_spliced_offset
Offset of last nucleotide in stop codon into the spliced mRNA (excluding introns)

name
Alias for transcript_name necessary for backward compatibility.

protein_id

protein_sequence

sequence
Spliced cDNA sequence of transcript (includes 5" UTR, coding sequence, and 3" UTR)

spliced_offset(*position*)
Convert from an absolute chromosomal position to the offset into this transcript"s spliced mRNA.

Position must be inside some exon (otherwise raise exception).

start_codon_positions
Chromosomal positions of nucleotides in start codon.

start_codon_spliced_offsets
Offsets from start of spliced mRNA transcript of nucleotides in start codon.

start_codon_unspliced_offsets
Offsets from start of unspliced pre-mRNA transcript of nucleotides in start codon.

stop_codon_positions
Chromosomal positions of nucleotides in stop codon.

stop_codon_spliced_offsets
Offsets from start of spliced mRNA transcript of nucleotides in stop codon.

stop_codon_unspliced_offsets
Offsets from start of unspliced pre-mRNA transcript of nucleotides in stop codon.

three_prime_utr_sequence
cDNA sequence of 3' UTR (untranslated region at the end of the transcript)

to_dict()
Returns a dictionary which can be used to reconstruct an instance of a derived class (typically by matching args to __init__). The values of the returned dictionary must be primitive atomic types (bool, string, int, float), primitive collections (int, list, tuple, set) or instances of Serializable.

The default implementation is to assume all the arguments to __init__ have fields of the same name on a serializable object.

1.1.20 Module contents

class pyensembl.**MemoryCache**
Bases: object

In-memory and on-disk caching of long-running queries and computations.

cached_dataframe(*csv_path*, *compute_fn*)
If a CSV path is in the _memory_cache, then return that cached value.

If we've already saved the DataFrame as a CSV then load it.

Otherwise run the provided *compute_fn*, and store its result in memory and and save it as a CSV.

cached_object(*path*, *compute_fn*)
If *cached_object* has already been called for a value of *path* in this running Python instance, then it should have a cached value in the

_memory_cache; return that value.

If this function was never called before with a particular value of *path*, then call compute_fn, and pickle it to *path*.

If *path* already exists, unpickle it and store that value in _memory_cache.

clear_cached_objects()

delete_file(*path*)

is_empty(*filename*)

remove_from_cache(*key*)

class pyensembl.**DownloadCache**(*reference_name*, *annotation_name*, *annotation_version=None*, *decompress_on_download=False*, *copy_local_files_to_cache=False*, *install_string_function=None*, *cache_directory_path=None*)

Bases: object

Downloads remote files to cache, optionally copies local files into cache, raises custom message if data is missing.

cache_directory_path

cached_path(*path_or_url*)

When downloading remote files, the default behavior is to name local files the same as their remote counterparts.

delete_cache_directory()

delete_cached_files(*prefixes=[]*, *suffixes=[]*)

Deletes any cached files matching the prefixes or suffixes given

download_or_copy_if_necessary(*path_or_url*, *download_if_missing=False*, *overwrite=False*)

Download a remote file or copy Get the local path to a possibly remote file.

Download if file is missing from the cache directory and *download_if_missing* is True. Download even if local file exists if both *download_if_missing* and *overwrite* are True.

If the file is on the local file system then return its path, unless self.copy_local_to_cache is True, and then copy it to the cache first.

path_or_url : str

download_if_missing [bool, optional] Download files if missing from local cache

overwrite [bool, optional] Overwrite existing copy if it exists

is_url_format(*path_or_url*)

Is the given string a URL?

path_or_url : str

bool

local_path_or_install_error(*field_name*, *path_or_url*, *download_if_missing=False*, *overwrite=False*)

class pyensembl.**Database**(*gtf_path*, *install_string=None*, *cache_directory_path=None*, *restrict_gtf_columns=None*, *restrict_gtf_features=None*)

Bases: object

Wrapper around sqlite3 database so that the rest of the library doesn't have to worry about constructing the .db file or writing SQL queries directly.

PRIMARY_KEY_COLUMNS = {'gene': 'gene_id', 'transcript': 'transcript_id'}

column_exists(*table_name*, *column_name*)

column_values_at_locus(*column_name*, *feature*, *contig*, *position*, *end=None*, *strand=None*, *distinct=False*, *sorted=False*)
Get the non-null values of a column from the database at a particular range of loci

columns(*table_name*)

connect_or_create(*overwrite=False*)
Return a connection to the database if it exists, otherwise create it. Overwrite the existing database if *overwrite* is True.

connection
Get a connection to the database or raise an exception

create(*overwrite=False*)
Create the local database (including indexing) if it's not already set up. If *overwrite* is True, always recreate the database from scratch.

Returns a connection to the database.

distinct_column_values_at_locus(*column*, *feature*, *contig*, *position*, *end=None*, *strand=None*)
Gather all the distinct values for a property/column at some specified locus.

column [str] Which property are we getting the values of.

feature [str] Which type of entry (e.g. transcript, exon, gene) is the property associated with?

contig [str] Chromosome or unplaced contig name

position [int] Chromosomal position

end [int, optional] End position of a range, if unspecified assume we're only looking at the single given position.

strand [str, optional] Either the positive ('+') or negative strand ('-'). If unspecified then check for values on either strand.

local_db_filename

local_db_path

query(***kwargs*)
Construct a SQL query and run against the sqlite3 database, filtered both by the feature type and a user-provided column/value.

query_distinct_on_contig(*column_name*, *feature*, *contig*)

query_feature_values(***kwargs*)
Run a SQL query against the sqlite3 database, filtered only on the feature type.

query_loci(*filter_column*, *filter_value*, *feature*)
Query for loci satisfying a given filter and feature type.

filter_column [str] Name of column to filter results by.

filter_value [str] Only return loci which have this value in the their filter_column.

feature [str] Feature names such as 'transcript', 'gene', and 'exon'

Returns list of Locus objects

query_locus(*filter_column*, *filter_value*, *feature*)
Query for unique locus, raises error if missing or more than one locus in the database.

filter_column [str] Name of column to filter results by.

filter_value [str] Only return loci which have this value in the their filter_column.

feature [str] Feature names such as 'transcript', 'gene', and 'exon'

Returns single Locus object.

query_one(*select_column_names*, *filter_column*, *filter_value*, *feature*, *distinct=False*, *required=False*)

run_sql_query(*sql*, *required=False*, *query_params=[]*)
Given an arbitrary SQL query, run it against the database and return the results.

sql [str] SQL query

required [bool] Raise an error if no results found in the database

query_params [list] For each '?' in the query there must be a corresponding value in this list.

class pyensembl.**EnsemblRelease**(*release=97, species=Species(latin_name='homo_sapiens', synonyms=['human'], reference_assemblies={'GRCh38': (76, 97), 'GRCh37': (55, 75), 'NCBI36': (54, 54)}), server='ftp://ftp.ensembl.org'*)
Bases: *pyensembl.genome.Genome*

Bundles together the genomic annotation and sequence data associated with a particular release of the Ensembl database.

classmethod cached(*release=97, species=Species(latin_name='homo_sapiens', synonyms=['human'], reference_assemblies={'GRCh38': (76, 97), 'GRCh37': (55, 75), 'NCBI36': (54, 54)}), server='ftp://ftp.ensembl.org'*)
Construct EnsemblRelease if it's never been made before, otherwise return an old instance.

classmethod from_dict(*state_dict*)
Deserialize EnsemblRelease without creating duplicate instances.

install_string()
Add every missing file to the install string shown to the user in an error message.

classmethod normalize_init_values(*release*, *species*, *server*)
Normalizes the arguments which uniquely specify an EnsemblRelease genome.

to_dict()
Returns a dictionary of the essential fields of this Genome.

pyensembl.**cached_release**(*release*, *species='human'*)
Create an EnsemblRelease instance only if it's hasn't already been made, otherwise returns the old instance. Keeping this function for backwards compatibility but this functionality has been moving into the cached method of EnsemblRelease.

class pyensembl.**Gene**(*gene_id*, *gene_name*, *contig*, *start*, *end*, *strand*, *biotype*, *genome*)
Bases: pyensembl.locus_with_genome.LocusWithGenome

exons

id
Alias for gene_id necessary for backwards compatibility.

name
Alias for gene_name necessary for backwards compatibility.

to_dict()
Returns a dictionary which can be used to reconstruct an instance of a derived class (typically by matching args to __init__). The values of the returned dictionary must be primitive atomic types (bool, string, int, float), primitive collections (int, list, tuple, set) or instances of Serializable.

The default implementation is to assume all the arguments to __init__ have fields of the same name on a serializable object.

transcripts
Property which dynamically construct transcript objects for all transcript IDs associated with this gene.

class pyensembl.**Transcript**(*transcript_id*, *transcript_name*, *contig*, *start*, *end*, *strand*, *biotype*, *gene_id*, *genome*, *support_level=None*)
Bases: pyensembl.locus_with_genome.LocusWithGenome

Transcript encompasses the locus, exons, and sequence of a transcript.

Lazily fetches sequence in case we"re constructing many Transcripts and not using the sequence, avoid the memory/performance overhead of fetching and storing sequences from a FASTA file.

coding_sequence
cDNA coding sequence (from start codon to stop codon, without any introns)

coding_sequence_position_ranges
Return absolute chromosome position ranges for CDS fragments of this transcript

complete
Consider a transcript complete if it has start and stop codons and a coding sequence whose length is divisible by 3

contains_start_codon
Does this transcript have an annotated start_codon entry?

contains_stop_codon
Does this transcript have an annotated stop_codon entry?

exon_intervals
List of (start,end) tuples for each exon of this transcript, in the order specified by the 'exon_number' column of the exon table.

exons

first_start_codon_spliced_offset
Offset of first nucleotide in start codon into the spliced mRNA (excluding introns)

five_prime_utr_sequence
cDNA sequence of 5' UTR (untranslated region at the beginning of the transcript)

gene

gene_name

id
Alias for transcript_id necessary for backward compatibility.

last_stop_codon_spliced_offset
Offset of last nucleotide in stop codon into the spliced mRNA (excluding introns)

name
Alias for transcript_name necessary for backward compatibility.

protein_id

protein_sequence

sequence
Spliced cDNA sequence of transcript (includes 5" UTR, coding sequence, and 3" UTR)

spliced_offset(*position*)
Convert from an absolute chromosomal position to the offset into this transcript"s spliced mRNA.

Position must be inside some exon (otherwise raise exception).

start_codon_positions
Chromosomal positions of nucleotides in start codon.

start_codon_spliced_offsets
Offsets from start of spliced mRNA transcript of nucleotides in start codon.

start_codon_unspliced_offsets
Offsets from start of unspliced pre-mRNA transcript of nucleotides in start codon.

stop_codon_positions
Chromosomal positions of nucleotides in stop codon.

stop_codon_spliced_offsets
Offsets from start of spliced mRNA transcript of nucleotides in stop codon.

stop_codon_unspliced_offsets
Offsets from start of unspliced pre-mRNA transcript of nucleotides in stop codon.

three_prime_utr_sequence
cDNA sequence of 3' UTR (untranslated region at the end of the transcript)

to_dict()
Returns a dictionary which can be used to reconstruct an instance of a derived class (typically by matching args to __init__). The values of the returned dictionary must be primitive atomic types (bool, string, int, float), primitive collections (int, list, tuple, set) or instances of Serializable.

The default implementation is to assume all the arguments to __init__ have fields of the same name on a serializable object.

class pyensembl.**Exon**(*exon_id*, *contig*, *start*, *end*, *strand*, *gene_name*, *gene_id*)
Bases: *pyensembl.locus.Locus*

id
Alias for exon_id necessary for backward compatibility.

to_dict()
Returns a dictionary which can be used to reconstruct an instance of a derived class (typically by matching args to __init__). The values of the returned dictionary must be primitive atomic types (bool, string, int, float), primitive collections (int, list, tuple, set) or instances of Serializable.

The default implementation is to assume all the arguments to __init__ have fields of the same name on a serializable object.

class pyensembl.**SequenceData**(*fasta_paths*, *cache_directory_path=None*)
Bases: object

Container for reference nucleotide and amino acid sequenes.

clear_cache()

fasta_dictionary

get(*sequence_id*)
Get sequence associated with given ID or return None if missing

index(*overwrite=False*)

pyensembl.**find_nearest_locus**(*start*, *end*, *loci*)
Finds nearest locus (object with method *distance_to_interval*) to the interval defined by the given *start* and *end* positions. Returns the distance to that locus, along with the locus object itself.

pyensembl.**find_species_by_name**(*species_name*)

pyensembl.**find_species_by_reference**(*reference_name*)

pyensembl.**genome_for_reference_name**(*reference_name*, *allow_older_downloaded_release=True*)
Given a genome reference name, such as "GRCh38", returns the corresponding Ensembl Release object.

If *allow_older_downloaded_release* is True, and some older releases have been downloaded, then return the most recent locally available release.

Otherwise, return the newest release of Ensembl (even if its data hasn't already been downloaded).

pyensembl.**which_reference**(*species_name*, *ensembl_release*)

pyensembl.**check_species_object**(*species_name_or_object*)
Helper for validating user supplied species names or objects.

pyensembl.**normalize_reference_name**(*name*)
Search the dictionary of species-specific references to find a reference name that matches aside from capitalization.

If no matching reference is found, raise an exception.

pyensembl.**normalize_species_name**(*name*)
If species name was "Homo sapiens" then replace spaces with underscores and return "homo_sapiens". Also replace common names like "human" with "homo_sapiens".

class pyensembl.**Genome**(*reference_name*, *annotation_name*, *annotation_version=None*, *gtf_path_or_url=None*, *transcript_fasta_paths_or_urls=None*, *protein_fasta_paths_or_urls=None*, *decompress_on_download=False*, *copy_local_files_to_cache=False*, *cache_directory_path=None*)
Bases: serializable.serializable.Serializable

Bundles together the genomic annotation and sequence data associated with a particular genomic database source (e.g. a single Ensembl release) and provides a wide variety of helper methods for accessing this data.

clear_cache()
Clear any in-memory cached values and short-lived on-disk materializations from MemoryCache

contigs()
Returns all contig names for any gene in the genome (field called "seqname" in Ensembl GTF files)

db

delete_index_files()
Delete all data aside from source GTF and FASTA files

download(*overwrite=False*)
Download data files needed by this Genome instance.

overwrite [bool, optional] Download files regardless whether local copy already exists.

exon_by_id(*exon_id*)
Construct an Exon object from its ID by looking up the exon"s properties in the given Database.

exon_ids(*contig=None*, *strand=None*)

exon_ids_at_locus(*contig*, *position*, *end=None*, *strand=None*)

exon_ids_of_gene_id(*gene_id*)

exon_ids_of_gene_name(*gene_name*)

exon_ids_of_transcript_id(*transcript_id*)

exon_ids_of_transcript_name(*transcript_name*)

exons(*contig=None*, *strand=None*)
Create exon object for all exons in the database, optionally restrict to a particular chromosome using the *contig* argument.

exons_at_locus(*contig*, *position*, *end=None*, *strand=None*)

gene_by_id(*gene_id*)
Construct a Gene object for the given gene ID.

gene_by_protein_id(*protein_id*)
Get the gene ID associated with the given protein ID, return its Gene object

gene_id_of_protein_id(*protein_id*)
What is the gene ID associated with a given protein ID?

gene_ids(*contig=None*, *strand=None*)
What are all the gene IDs (optionally restrict to a given chromosome/contig and/or strand)

gene_ids_at_locus(*contig*, *position*, *end=None*, *strand=None*)

gene_ids_of_gene_name(*gene_name*)
What are the gene IDs associated with a given gene name? (due to copy events, there might be multiple genes per name)

gene_name_of_exon_id(*exon_id*)

gene_name_of_gene_id(*gene_id*)

gene_name_of_transcript_id(*transcript_id*)

gene_name_of_transcript_name(*transcript_name*)

gene_names(*contig=None*, *strand=None*)
Return all genes in the database, optionally restrict to a chromosome and/or strand.

gene_names_at_locus(*contig*, *position*, *end=None*, *strand=None*)

genes(*contig=None*, *strand=None*)
Returns all Gene objects in the database. Can be restricted to a particular contig/chromosome and strand by the following arguments:

contig [str] Only return genes on the given contig.

strand [str] Only return genes on this strand.

genes_at_locus(*contig*, *position*, *end=None*, *strand=None*)

genes_by_name(*gene_name*)
Get all the unqiue genes with the given name (there might be multiple due to copies in the genome), return a list containing a Gene object for each distinct ID.

index(*overwrite=False*)
Assuming that all necessary data for this Genome has been downloaded, generate the GTF database and save efficient representation of FASTA sequence files.

install_string()
Add every missing file to the install string shown to the user in an error message.

loci_of_gene_names(*gene_name*)

Given a gene name returns list of Locus objects with fields: chromosome, start, stop, strand

You can get multiple results since a gene might have multiple copies in the genome.

locus_of_exon_id(*exon_id*)
Given an exon ID returns Locus

locus_of_gene_id(*gene_id*)
Given a gene ID returns Locus with: chromosome, start, stop, strand

locus_of_transcript_id(*transcript_id*)

protein_ids(*contig=None*, *strand=None*)
What are all the protein IDs (optionally restrict to a given chromosome and/or strand)

protein_ids_at_locus(*contig*, *position*, *end=None*, *strand=None*)

protein_sequence(*protein_id*)
Return cDNA nucleotide sequence of transcript, or None if transcript doesn't have cDNA sequence.

protein_sequences

required_local_files()

required_local_files_exist(*empty_files_ok=False*)

requires_gtf

requires_protein_fasta

requires_transcript_fasta

to_dict()
Returns a dictionary of the essential fields of this Genome.

transcript_by_id(*transcript_id*)
Construct Transcript object with given transcript ID

transcript_by_protein_id(*protein_id*)

transcript_id_of_protein_id(*protein_id*)
What is the transcript ID associated with a given protein ID?

transcript_ids(*contig=None*, *strand=None*)

transcript_ids_at_locus(*contig*, *position*, *end=None*, *strand=None*)

transcript_ids_of_exon_id(*exon_id*)

transcript_ids_of_gene_id(*gene_id*)

transcript_ids_of_gene_name(*gene_name*)

transcript_ids_of_transcript_name(*transcript_name*)

transcript_name_of_transcript_id(*transcript_id*)

transcript_names(*contig=None*, *strand=None*)
What are all the transcript names in the database (optionally, restrict to a given chromosome and/or strand)

transcript_names_at_locus(*contig*, *position*, *end=None*, *strand=None*)

transcript_names_of_gene_name(*gene_name*)

transcript_sequence(*transcript_id*)
Return cDNA nucleotide sequence of transcript, or None if transcript doesn't have cDNA sequence.

transcript_sequences

transcripts(*contig=None*, *strand=None*)
Construct Transcript object for every transcript entry in the database. Optionally restrict to a particular chromosome using the *contig* argument.

transcripts_at_locus(*contig*, *position*, *end=None*, *strand=None*)

transcripts_by_name(*transcript_name*)

class pyensembl.**Locus**(*contig*, *start*, *end*, *strand*)
Bases: serializable.serializable.Serializable

Base class for any entity which can be localized at a range of positions on a particular strand of a chromosome/contig.

can_overlap(*contig*, *strand=None*)
Is this locus on the same contig and (optionally) on the same strand?

contains(*contig*, *start*, *end*, *strand=None*)

contains_locus(*other_locus*)

distance_to_interval(*start*, *end*)
Find the distance between intervals [start1, end1] and [start2, end2]. If the intervals overlap then the distance is 0.

distance_to_locus(*other*)

length

offset(*position*)
Offset of given position from stranded start of this locus.

For example, if a Locus goes from 10..20 and is on the negative strand, then the offset of position 13 is 7, whereas if the Locus is on the positive strand, then the offset is 3.

offset_range(*start*, *end*)
Database start/end entries are always ordered such that start < end. This makes computing a relative position (e.g. of a stop codon relative to its transcript) complicated since the "end" position of a backwards locus is actually earlir on the strand. This function correctly selects a start vs. end value depending on this locuses's strand and determines that position's offset from the earliest position in this locus.

on_backward_strand

on_contig(*contig*)

on_forward_strand

on_negative_strand

on_positive_strand

on_strand(*strand*)

overlaps(*contig*, *start*, *end*, *strand=None*)
Does this locus overlap with a given range of positions?

Since locus position ranges are inclusive, we should make sure that e.g. chr1:10-10 overlaps with chr1:10-10

overlaps_locus(*other_locus*)

to_dict()
Returns a dictionary which can be used to reconstruct an instance of a derived class (typically by matching args to __init__). The values of the returned dictionary must be primitive atomic types (bool, string, int, float), primitive collections (int, list, tuple, set) or instances of Serializable.

The default implementation is to assume all the arguments to __init__ have fields of the same name on a serializable object.

to_tuple()

class pyensembl.**Exon**(*exon_id*, *contig*, *start*, *end*, *strand*, *gene_name*, *gene_id*)

Bases: *pyensembl.locus.Locus*

id

Alias for exon_id necessary for backward compatibility.

to_dict()

Returns a dictionary which can be used to reconstruct an instance of a derived class (typically by matching args to __init__). The values of the returned dictionary must be primitive atomic types (bool, string, int, float), primitive collections (int, list, tuple, set) or instances of Serializable.

The default implementation is to assume all the arguments to __init__ have fields of the same name on a serializable object.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

pyensembl, 15
pyensembl.common, 3
pyensembl.database, 3
pyensembl.download_cache, 5
pyensembl.ensembl_release, 6
pyensembl.ensembl_release_versions, 6
pyensembl.ensembl_url_templates, 6
pyensembl.exon, 7
pyensembl.gene, 7
pyensembl.genome, 8
pyensembl.locus, 11
pyensembl.memory_cache, 12
pyensembl.search, 12
pyensembl.sequence_data, 12
pyensembl.shell, 13
pyensembl.species, 13
pyensembl.transcript, 14

Index

A

all_combinations_of_ensembl_genomes() (*in module pyensembl.shell*), 13
all_registered_latin_names() (*pyensembl.species.Species class method*), 13
all_species_release_pairs() (*pyensembl.species.Species class method*), 13

C

cache_directory_path (*pyensembl.download_cache.DownloadCache attribute*), 5
cache_directory_path (*pyensembl.DownloadCache attribute*), 16
cache_subdirectory() (*in module pyensembl.download_cache*), 6
cached() (*pyensembl.ensembl_release.EnsemblRelease class method*), 6
cached() (*pyensembl.EnsemblRelease class method*), 18
cached_dataframe() (*pyensembl.memory_cache.MemoryCache method*), 12
cached_dataframe() (*pyensembl.MemoryCache method*), 15
cached_object() (*pyensembl.memory_cache.MemoryCache method*), 12
cached_object() (*pyensembl.MemoryCache method*), 15
cached_path() (*pyensembl.download_cache.DownloadCache method*), 5
cached_path() (*pyensembl.DownloadCache method*), 16
cached_release() (*in module pyensembl*), 18
cached_release() (*in module pyensembl.ensembl_release*), 6
can_overlap() (*pyensembl.Locus method*), 24
can_overlap() (*pyensembl.locus.Locus method*), 11
check_release_number() (*in module pyensembl.ensembl_release_versions*), 6
check_species_object() (*in module pyensembl*), 21
check_species_object() (*in module pyensembl.species*), 14
clear_cache() (*pyensembl.Genome method*), 21
clear_cache() (*pyensembl.genome.Genome method*), 8
clear_cache() (*pyensembl.sequence_data.SequenceData method*), 12
clear_cache() (*pyensembl.SequenceData method*), 20
clear_cached_objects() (*pyensembl.memory_cache.MemoryCache method*), 12
clear_cached_objects() (*pyensembl.MemoryCache method*), 16
coding_sequence (*pyensembl.Transcript attribute*), 19
coding_sequence (*pyensembl.transcript.Transcript attribute*), 14
coding_sequence_position_ranges (*pyensembl.Transcript attribute*), 19
coding_sequence_position_ranges (*pyensembl.transcript.Transcript attribute*), 14
collect_all_installed_ensembl_releases() (*in module pyensembl.shell*), 13
collect_selected_genomes() (*in module pyensembl.shell*), 13
column_exists() (*pyensembl.Database method*), 16
column_exists() (*pyensembl.database.Database method*), 3
column_values_at_locus() (*pyensembl.Database method*), 17
column_values_at_locus() (*pyensembl.database.Database method*), 3

columns() (*pyensembl.Database method*), 17
columns() (*pyensembl.database.Database method*), 3
complete (*pyensembl.Transcript attribute*), 19
complete (*pyensembl.transcript.Transcript attribute*), 14
connect_or_create() (*pyensembl.Database method*), 17
connect_or_create() (*pyensembl.database.Database method*), 4
connection (*pyensembl.Database attribute*), 17
connection (*pyensembl.database.Database attribute*), 4
contains() (*pyensembl.Locus method*), 24
contains() (*pyensembl.locus.Locus method*), 11
contains_locus() (*pyensembl.Locus method*), 24
contains_locus() (*pyensembl.locus.Locus method*), 11
contains_start_codon (*pyensembl.Transcript attribute*), 19
contains_start_codon (*pyensembl.transcript.Transcript attribute*), 14
contains_stop_codon (*pyensembl.Transcript attribute*), 19
contains_stop_codon (*pyensembl.transcript.Transcript attribute*), 14
contigs() (*pyensembl.Genome method*), 21
contigs() (*pyensembl.genome.Genome method*), 8
create() (*pyensembl.Database method*), 17
create() (*pyensembl.database.Database method*), 4

D

Database (*class in pyensembl*), 16
Database (*class in pyensembl.database*), 3
db (*pyensembl.Genome attribute*), 21
db (*pyensembl.genome.Genome attribute*), 8
delete_cache_directory() (*pyensembl.download_cache.DownloadCache method*), 5
delete_cache_directory() (*pyensembl.DownloadCache method*), 16
delete_cached_files() (*pyensembl.download_cache.DownloadCache method*), 5
delete_cached_files() (*pyensembl.DownloadCache method*), 16
delete_file() (*pyensembl.memory_cache.MemoryCache method*), 12
delete_file() (*pyensembl.MemoryCache method*), 16
delete_index_files() (*pyensembl.Genome method*), 21
delete_index_files() (*pyensembl.genome.Genome method*), 8
distance_to_interval() (*pyensembl.Locus method*), 24
distance_to_interval() (*pyensembl.locus.Locus method*), 11
distance_to_locus() (*pyensembl.Locus method*), 24
distance_to_locus() (*pyensembl.locus.Locus method*), 11
distinct_column_values_at_locus() (*pyensembl.Database method*), 17
distinct_column_values_at_locus() (*pyensembl.database.Database method*), 4
download() (*pyensembl.Genome method*), 21
download() (*pyensembl.genome.Genome method*), 8
download_or_copy_if_necessary() (*pyensembl.download_cache.DownloadCache method*), 5
download_or_copy_if_necessary() (*pyensembl.DownloadCache method*), 16
DownloadCache (*class in pyensembl*), 16
DownloadCache (*class in pyensembl.download_cache*), 5
dump_pickle() (*in module pyensembl.common*), 3

E

EnsemblRelease (*class in pyensembl*), 18
EnsemblRelease (*class in pyensembl.ensembl_release*), 6
Exon (*class in pyensembl*), 20, 24
Exon (*class in pyensembl.exon*), 7
exon_by_id() (*pyensembl.Genome method*), 21
exon_by_id() (*pyensembl.genome.Genome method*), 8
exon_ids() (*pyensembl.Genome method*), 21
exon_ids() (*pyensembl.genome.Genome method*), 8
exon_ids_at_locus() (*pyensembl.Genome method*), 21
exon_ids_at_locus() (*pyensembl.genome.Genome method*), 8
exon_ids_of_gene_id() (*pyensembl.Genome method*), 21
exon_ids_of_gene_id() (*pyensembl.genome.Genome method*), 8
exon_ids_of_gene_name() (*pyensembl.Genome method*), 21
exon_ids_of_gene_name() (*pyensembl.genome.Genome method*), 8
exon_ids_of_transcript_id() (*pyensembl.Genome method*), 21
exon_ids_of_transcript_id() (*pyensembl.genome.Genome method*), 8

exon_ids_of_transcript_name() (*pyensembl.Genome method*), 21
exon_ids_of_transcript_name() (*pyensembl.genome.Genome method*), 8
exon_intervals (*pyensembl.Transcript attribute*), 19
exon_intervals (*pyensembl.transcript.Transcript attribute*), 14
exons (*pyensembl.Gene attribute*), 18
exons (*pyensembl.gene.Gene attribute*), 7
exons (*pyensembl.Transcript attribute*), 19
exons (*pyensembl.transcript.Transcript attribute*), 14
exons() (*pyensembl.Genome method*), 21
exons() (*pyensembl.genome.Genome method*), 8
exons_at_locus() (*pyensembl.Genome method*), 22
exons_at_locus() (*pyensembl.genome.Genome method*), 8

F

fasta_dictionary (*pyensembl.sequence_data.SequenceData attribute*), 12
fasta_dictionary (*pyensembl.SequenceData attribute*), 20
find_nearest_locus() (*in module pyensembl*), 20
find_nearest_locus() (*in module pyensembl.search*), 12
find_species_by_name() (*in module pyensembl*), 20
find_species_by_name() (*in module pyensembl.species*), 14
find_species_by_reference() (*in module pyensembl*), 20
first_start_codon_spliced_offset (*pyensembl.Transcript attribute*), 19
first_start_codon_spliced_offset (*pyensembl.transcript.Transcript attribute*), 14
five_prime_utr_sequence (*pyensembl.Transcript attribute*), 19
five_prime_utr_sequence (*pyensembl.transcript.Transcript attribute*), 14
from_dict() (*pyensembl.ensembl_release.EnsemblRelease class method*), 6
from_dict() (*pyensembl.EnsemblRelease class method*), 18
from_dict() (*pyensembl.species.Species class method*), 13

G

Gene (*class in pyensembl*), 18
Gene (*class in pyensembl.gene*), 7
gene (*pyensembl.Transcript attribute*), 19
gene (*pyensembl.transcript.Transcript attribute*), 14
gene_by_id() (*pyensembl.Genome method*), 22
gene_by_id() (*pyensembl.genome.Genome method*), 8
gene_by_protein_id() (*pyensembl.Genome method*), 22
gene_by_protein_id() (*pyensembl.genome.Genome method*), 8
gene_id_of_protein_id() (*pyensembl.Genome method*), 22
gene_id_of_protein_id() (*pyensembl.genome.Genome method*), 8
gene_ids() (*pyensembl.Genome method*), 22
gene_ids() (*pyensembl.genome.Genome method*), 9
gene_ids_at_locus() (*pyensembl.Genome method*), 22
gene_ids_at_locus() (*pyensembl.genome.Genome method*), 9
gene_ids_of_gene_name() (*pyensembl.Genome method*), 22
gene_ids_of_gene_name() (*pyensembl.genome.Genome method*), 9
gene_name (*pyensembl.Transcript attribute*), 19
gene_name (*pyensembl.transcript.Transcript attribute*), 14
gene_name_of_exon_id() (*pyensembl.Genome method*), 22
gene_name_of_exon_id() (*pyensembl.genome.Genome method*), 9
gene_name_of_gene_id() (*pyensembl.Genome method*), 22
gene_name_of_gene_id() (*pyensembl.genome.Genome method*), 9
gene_name_of_transcript_id() (*pyensembl.Genome method*), 22
gene_name_of_transcript_id() (*pyensembl.genome.Genome method*), 9
gene_name_of_transcript_name() (*pyensembl.Genome method*), 22
gene_name_of_transcript_name() (*pyensembl.genome.Genome method*), 9
gene_names() (*pyensembl.Genome method*), 22
gene_names() (*pyensembl.genome.Genome method*), 9
gene_names_at_locus() (*pyensembl.Genome method*), 22
gene_names_at_locus() (*pyensembl.genome.Genome method*), 9
genes() (*pyensembl.Genome method*), 22
genes() (*pyensembl.genome.Genome method*), 9
genes_at_locus() (*pyensembl.Genome method*), 22
genes_at_locus() (*pyensembl.genome.Genome method*), 9
genes_by_name() (*pyensembl.Genome method*), 22
genes_by_name() (*pyensembl.genome.Genome method*), 9

Genome (*class in pyensembl*), 21
Genome (*class in pyensembl.genome*), 8
genome_for_reference_name() (*in module pyensembl*), 21
get() (*pyensembl.sequence_data.SequenceData method*), 12
get() (*pyensembl.SequenceData method*), 20

I

id (*pyensembl.Exon attribute*), 20, 25
id (*pyensembl.exon.Exon attribute*), 7
id (*pyensembl.Gene attribute*), 18
id (*pyensembl.gene.Gene attribute*), 7
id (*pyensembl.Transcript attribute*), 19
id (*pyensembl.transcript.Transcript attribute*), 14
index() (*pyensembl.Genome method*), 22
index() (*pyensembl.genome.Genome method*), 9
index() (*pyensembl.sequence_data.SequenceData method*), 12
index() (*pyensembl.SequenceData method*), 20
install_string() (*pyensembl.ensembl_release.EnsemblRelease method*), 6
install_string() (*pyensembl.EnsemblRelease method*), 18
install_string() (*pyensembl.Genome method*), 22
install_string() (*pyensembl.genome.Genome method*), 9
is_empty() (*pyensembl.memory_cache.MemoryCache method*), 12
is_empty() (*pyensembl.MemoryCache method*), 16
is_url_format() (*pyensembl.download_cache.DownloadCache method*), 5
is_url_format() (*pyensembl.DownloadCache method*), 16

L

last_stop_codon_spliced_offset (*pyensembl.Transcript attribute*), 19
last_stop_codon_spliced_offset (*pyensembl.transcript.Transcript attribute*), 14
length (*pyensembl.Locus attribute*), 24
length (*pyensembl.locus.Locus attribute*), 11
load_pickle() (*in module pyensembl.common*), 3
local_db_filename (*pyensembl.Database attribute*), 17
local_db_filename (*pyensembl.database.Database attribute*), 4
local_db_path (*pyensembl.Database attribute*), 17
local_db_path (*pyensembl.database.Database attribute*), 4
local_path_or_install_error() (*pyensembl.download_cache.DownloadCache method*), 5
local_path_or_install_error() (*pyensembl.DownloadCache method*), 16
loci_of_gene_names() (*pyensembl.Genome method*), 22
loci_of_gene_names() (*pyensembl.genome.Genome method*), 9
Locus (*class in pyensembl*), 24
Locus (*class in pyensembl.locus*), 11
locus_of_exon_id() (*pyensembl.Genome method*), 22
locus_of_exon_id() (*pyensembl.genome.Genome method*), 9
locus_of_gene_id() (*pyensembl.Genome method*), 22
locus_of_gene_id() (*pyensembl.genome.Genome method*), 9
locus_of_transcript_id() (*pyensembl.Genome method*), 23
locus_of_transcript_id() (*pyensembl.genome.Genome method*), 9

M

make_fasta_filename() (*in module pyensembl.ensembl_url_templates*), 7
make_fasta_url() (*in module pyensembl.ensembl_url_templates*), 7
make_gtf_filename() (*in module pyensembl.ensembl_url_templates*), 7
make_gtf_url() (*in module pyensembl.ensembl_url_templates*), 7
memoize() (*in module pyensembl.common*), 3
MemoryCache (*class in pyensembl*), 15
MemoryCache (*class in pyensembl.memory_cache*), 12
MissingLocalFile, 5
MissingRemoteFile, 5

N

name (*pyensembl.Gene attribute*), 18
name (*pyensembl.gene.Gene attribute*), 7
name (*pyensembl.Transcript attribute*), 19
name (*pyensembl.transcript.Transcript attribute*), 14
normalize_init_values() (*pyensembl.ensembl_release.EnsemblRelease class method*), 6
normalize_init_values() (*pyensembl.EnsemblRelease class method*), 18
normalize_reference_name() (*in module pyensembl*), 21
normalize_release_properties() (*in module pyensembl.ensembl_url_templates*), 7
normalize_species_name() (*in module pyensembl*), 21
normalize_species_name() (*in module pyensembl.species*), 14

O

offset() (*pyensembl.Locus method*), 24
offset() (*pyensembl.locus.Locus method*), 11
offset_range() (*pyensembl.Locus method*), 24
offset_range() (*pyensembl.locus.Locus method*), 11
on_backward_strand (*pyensembl.Locus attribute*), 24
on_backward_strand (*pyensembl.locus.Locus attribute*), 11
on_contig() (*pyensembl.Locus method*), 24
on_contig() (*pyensembl.locus.Locus method*), 11
on_forward_strand (*pyensembl.Locus attribute*), 24
on_forward_strand (*pyensembl.locus.Locus attribute*), 11
on_negative_strand (*pyensembl.Locus attribute*), 24
on_negative_strand (*pyensembl.locus.Locus attribute*), 11
on_positive_strand (*pyensembl.Locus attribute*), 24
on_positive_strand (*pyensembl.locus.Locus attribute*), 11
on_strand() (*pyensembl.Locus method*), 24
on_strand() (*pyensembl.locus.Locus method*), 11
overlaps() (*pyensembl.Locus method*), 24
overlaps() (*pyensembl.locus.Locus method*), 11
overlaps_locus() (*pyensembl.Locus method*), 24
overlaps_locus() (*pyensembl.locus.Locus method*), 11

P

PRIMARY_KEY_COLUMNS (*pyensembl.Database attribute*), 16
PRIMARY_KEY_COLUMNS (*pyensembl.database.Database attribute*), 3
protein_id (*pyensembl.Transcript attribute*), 19
protein_id (*pyensembl.transcript.Transcript attribute*), 14
protein_ids() (*pyensembl.Genome method*), 23
protein_ids() (*pyensembl.genome.Genome method*), 9
protein_ids_at_locus() (*pyensembl.Genome method*), 23
protein_ids_at_locus() (*pyensembl.genome.Genome method*), 9
protein_sequence (*pyensembl.Transcript attribute*), 19
protein_sequence (*pyensembl.transcript.Transcript attribute*), 15
protein_sequence() (*pyensembl.Genome method*), 23
protein_sequence() (*pyensembl.genome.Genome method*), 9
protein_sequences (*pyensembl.Genome attribute*), 23
protein_sequences (*pyensembl.genome.Genome attribute*), 10
pyensembl (*module*), 15
pyensembl.common (*module*), 3
pyensembl.database (*module*), 3
pyensembl.download_cache (*module*), 5
pyensembl.ensembl_release (*module*), 6
pyensembl.ensembl_release_versions (*module*), 6
pyensembl.ensembl_url_templates (*module*), 6
pyensembl.exon (*module*), 7
pyensembl.gene (*module*), 7
pyensembl.genome (*module*), 8
pyensembl.locus (*module*), 11
pyensembl.memory_cache (*module*), 12
pyensembl.search (*module*), 12
pyensembl.sequence_data (*module*), 12
pyensembl.shell (*module*), 13
pyensembl.species (*module*), 13
pyensembl.transcript (*module*), 14

Q

query() (*pyensembl.Database method*), 17
query() (*pyensembl.database.Database method*), 4
query_distinct_on_contig() (*pyensembl.Database method*), 17
query_distinct_on_contig() (*pyensembl.database.Database method*), 4
query_feature_values() (*pyensembl.Database method*), 17
query_feature_values() (*pyensembl.database.Database method*), 4
query_loci() (*pyensembl.Database method*), 17
query_loci() (*pyensembl.database.Database method*), 4
query_locus() (*pyensembl.Database method*), 17
query_locus() (*pyensembl.database.Database method*), 4
query_one() (*pyensembl.Database method*), 18
query_one() (*pyensembl.database.Database method*), 4

R

register() (*pyensembl.species.Species class method*), 13
remove_from_cache() (*pyensembl.memory_cache.MemoryCache*

method), 12
remove_from_cache() (*pyensembl.MemoryCache method*), 16
required_local_files() (*pyensembl.Genome method*), 23
required_local_files() (*pyensembl.genome.Genome method*), 10
required_local_files_exist() (*pyensembl.Genome method*), 23
required_local_files_exist() (*pyensembl.genome.Genome method*), 10
requires_gtf (*pyensembl.Genome attribute*), 23
requires_gtf (*pyensembl.genome.Genome attribute*), 10
requires_protein_fasta (*pyensembl.Genome attribute*), 23
requires_protein_fasta (*pyensembl.genome.Genome attribute*), 10
requires_transcript_fasta (*pyensembl.Genome attribute*), 23
requires_transcript_fasta (*pyensembl.genome.Genome attribute*), 10
run() (*in module pyensembl.shell*), 13
run_sql_query() (*pyensembl.Database method*), 18
run_sql_query() (*pyensembl.database.Database method*), 5

S

sequence (*pyensembl.Transcript attribute*), 19
sequence (*pyensembl.transcript.Transcript attribute*), 15
SequenceData (*class in pyensembl*), 20
SequenceData (*class in pyensembl.sequence_data*), 12
Species (*class in pyensembl.species*), 13
spliced_offset() (*pyensembl.Transcript method*), 19
spliced_offset() (*pyensembl.transcript.Transcript method*), 15
start_codon_positions (*pyensembl.Transcript attribute*), 20
start_codon_positions (*pyensembl.transcript.Transcript attribute*), 15
start_codon_spliced_offsets (*pyensembl.Transcript attribute*), 20
start_codon_spliced_offsets (*pyensembl.transcript.Transcript attribute*), 15
start_codon_unspliced_offsets (*pyensembl.Transcript attribute*), 20
start_codon_unspliced_offsets (*pyensembl.transcript.Transcript attribute*), 15
stop_codon_positions (*pyensembl.Transcript attribute*), 20
stop_codon_positions (*pyensembl.transcript.Transcript attribute*), 15
stop_codon_spliced_offsets (*pyensembl.Transcript attribute*), 20
stop_codon_spliced_offsets (*pyensembl.transcript.Transcript attribute*), 15
stop_codon_unspliced_offsets (*pyensembl.Transcript attribute*), 20
stop_codon_unspliced_offsets (*pyensembl.transcript.Transcript attribute*), 15

T

three_prime_utr_sequence (*pyensembl.Transcript attribute*), 20
three_prime_utr_sequence (*pyensembl.transcript.Transcript attribute*), 15
to_dict() (*pyensembl.ensembl_release.EnsemblRelease method*), 6
to_dict() (*pyensembl.EnsemblRelease method*), 18
to_dict() (*pyensembl.Exon method*), 20, 25
to_dict() (*pyensembl.exon.Exon method*), 7
to_dict() (*pyensembl.Gene method*), 18
to_dict() (*pyensembl.gene.Gene method*), 7
to_dict() (*pyensembl.Genome method*), 23
to_dict() (*pyensembl.genome.Genome method*), 10
to_dict() (*pyensembl.Locus method*), 24
to_dict() (*pyensembl.locus.Locus method*), 11
to_dict() (*pyensembl.species.Species method*), 13
to_dict() (*pyensembl.Transcript method*), 20
to_dict() (*pyensembl.transcript.Transcript method*), 15
to_tuple() (*pyensembl.Locus method*), 24
to_tuple() (*pyensembl.locus.Locus method*), 11
Transcript (*class in pyensembl*), 19
Transcript (*class in pyensembl.transcript*), 14
transcript_by_id() (*pyensembl.Genome method*), 23
transcript_by_id() (*pyensembl.genome.Genome method*), 10
transcript_by_protein_id() (*pyensembl.Genome method*), 23
transcript_by_protein_id() (*pyensembl.genome.Genome method*), 10
transcript_id_of_protein_id() (*pyensembl.Genome method*), 23
transcript_id_of_protein_id() (*pyensembl.genome.Genome method*), 10
transcript_ids() (*pyensembl.Genome method*), 23

transcript_ids() (*pyensembl.genome.Genome method*), 10
transcript_ids_at_locus() (*pyensembl.Genome method*), 23
transcript_ids_at_locus() (*pyensembl.genome.Genome method*), 10
transcript_ids_of_exon_id() (*pyensembl.Genome method*), 23
transcript_ids_of_exon_id() (*pyensembl.genome.Genome method*), 10
transcript_ids_of_gene_id() (*pyensembl.Genome method*), 23
transcript_ids_of_gene_id() (*pyensembl.genome.Genome method*), 10
transcript_ids_of_gene_name() (*pyensembl.Genome method*), 23
transcript_ids_of_gene_name() (*pyensembl.genome.Genome method*), 10
transcript_ids_of_transcript_name() (*pyensembl.Genome method*), 23
transcript_ids_of_transcript_name() (*pyensembl.genome.Genome method*), 10
transcript_name_of_transcript_id() (*pyensembl.Genome method*), 23
transcript_name_of_transcript_id() (*pyensembl.genome.Genome method*), 10
transcript_names() (*pyensembl.Genome method*), 23
transcript_names() (*pyensembl.genome.Genome method*), 10
transcript_names_at_locus() (*pyensembl.Genome method*), 23
transcript_names_at_locus() (*pyensembl.genome.Genome method*), 10
transcript_names_of_gene_name() (*pyensembl.Genome method*), 23
transcript_names_of_gene_name() (*pyensembl.genome.Genome method*), 10
transcript_sequence() (*pyensembl.Genome method*), 23
transcript_sequence() (*pyensembl.genome.Genome method*), 10
transcript_sequences (*pyensembl.Genome attribute*), 23
transcript_sequences (*pyensembl.genome.Genome attribute*), 10
transcripts (*pyensembl.Gene attribute*), 19
transcripts (*pyensembl.gene.Gene attribute*), 8
transcripts() (*pyensembl.Genome method*), 23
transcripts() (*pyensembl.genome.Genome method*), 10
transcripts_at_locus() (*pyensembl.Genome method*), 23
transcripts_at_locus() (*pyensembl.genome.Genome method*), 10
transcripts_by_name() (*pyensembl.Genome method*), 23
transcripts_by_name() (*pyensembl.genome.Genome method*), 10

W

which_reference() (*in module pyensembl*), 21
which_reference() (*pyensembl.species.Species method*), 13