SeqAnn Documentation

Release 1.0.0

Mike Halagan

Nov 12, 2019

User Documentation

Python package for annotating gene features

- Free software: LGPL 3.0
- Documentation: https://seqann.readthedocs.io.
- Jupyter Notebook

CHAPTER 1

Overview

The `seqann` package allows users to annotate gene features in consensus sequences. Annotations can be created by passing consensus sequences to the `annotate` method in the `BioSeqAnn` class. No parameters are required when initalizing a `BioSeqAnn` class. However, annotations can be created significantly faster when using a BioSQL database. When a BioSQL database is not provided the lastest hla.dat file is downloaded and parsed. A BioSQL database containing all of IPD-IMGT/HLA is available on DockerHub and can be run on any machine that has docker installed.

CHAPTER 2

Install

```
pip install seq-ann
```

CHAPTER 3

Parameters

Below are the list of parameters and the default values used when initalizing a `BioSeqAnn` object.

Parameter	Type	Default	Description
server	`BioSeqDatabase`	None	A BioSQL database containing all of the sequence data from IPD-IMGT/HLA.
dbversion	`str`	Latest	The IPD-IMGT/HLA or KIR database release.
datfile	`str`	None	The IPD-IMGT/HLA or KIR dat file to use in place of the **server** parameter.
kir	`bool`	False	Flag for indicating the input sequences are from the KIR gene system.
align	`bool`	False	Flag for producing the alignments along with the annotations.
verbose	`bool`	False	Flag for running in verbose mode.
verbosity	`int`	None	Numerical value to indicate how verbose the output will be in verbose mode.
debug	`Dict`	None	A dictionary containing a process names as the key and verbosity as the value

CHAPTER 4

Usage

To annotated a sequence initialize a new `BioSeqAnn` object and then pass the sequence to the `annotate` method. The sequence must be a Biopython `Seq`. The locus of the sequence is not required but it will improve the accuracy of the annotation.

The packages *ncbi-blast+* and *clustalo* are required to be installed on your system.

Set variables to BioSQL host/port if using BioSQL.

```
export BIOSQLHOST="localhost"
export BIOSQLPORT=3306
```

```
from seqann import BioSeqAnn
seqann = BioSeqAnn()
ann = seqann.annotate(sequence, "HLA-A")
```

The annotation of sequence can be done with or without providing a `BioSeqDatabase`. To use a BioSQL database initialize a `BioSeqDatabase` with the parameters that match the database you have running. If you are running the imgt_biosqldb from DockerHub then the following parameters we be the same.

```
from seqann import BioSeqAnn
from BioSQL import BioSeqDatabase
server = BioSeqDatabase.open_database(driver="pymysql", user="root",
                                      passwd="my-secret-pw", host="localhost",
                                      db="bioseqdb", port=3306)
seqann = BioSeqAnn(server=server)
ann = seqann.annotate(sequence, "HLA-A")
```

You may need to set environment variables: *BIOSQLHOST* (e.g. "localhost") and *BIOSQLPORT* (e.g. 3306) to your docker instance.

CHAPTER 5

Annotations

```
{
     'complete_annotation': True,
     'annotation': {'exon_1': SeqRecord(seq=Seq('AGAGACTCTCCCG',␣
↪SingleLetterAlphabet()), id='HLA:HLA00630', name='HLA:HLA00630', description=
↪'HLA:HLA00630 DQB1*03:04:01 597 bp', dbxrefs=[]),
                    'exon_2': SeqRecord(seq=Seq(
↪'AGGATTTCGTGTACCAGTTTAAGGCCATGTGCTACTTCACCAACGGGACGGAGC...GAG',␣
↪SingleLetterAlphabet()), id='HLA:HLA00630', name='HLA:HLA00630', description=
↪'HLA:HLA00630 DQB1*03:04:01 597 bp', dbxrefs=[]),
                    'exon_3': SeqRecord(seq=Seq(
↪'TGGAGCCCACAGTGACCATCTCCCCATCCAGGACAGAGGCCCTCAACCACCACA...ATG',␣
↪SingleLetterAlphabet()), id='HLA:HLA00630', name='<unknown name>', description=
↪'HLA:HLA00630', dbxrefs=[])},
     'features': {'exon_1': SeqFeature(FeatureLocation(ExactPosition(0),␣
↪ExactPosition(13), strand=1), type='exon_1'),
                  'exon_2': SeqFeature(FeatureLocation(ExactPosition(13),␣
↪ExactPosition(283), strand=1), type='exon_2')
                  'exon_3': SeqFeature(FeatureLocation(ExactPosition(283),␣
↪ExactPosition(503), strand=1), type='exon_3')},
     'method': 'nt_search and clustalo',
     'gfe': 'HLA-Aw2-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-4',
     'seq': SeqRecord(seq=Seq('AGAGACTCTCCCGAGGATTTCGTGTACCAGTTTAAGGCCATGTGCTACTTCACC.
↪..ATG', SingleLetterAlphabet()), id='HLA:HLA00630', name='HLA:HLA00630',␣
↪description='HLA:HLA00630 DQB1*03:04:01 597 bp', dbxrefs=[])
}
```

Once a sequence has been annotated the gene features and their corresponding sequences are available in the returned `Annotation` object. If a full annotation is not able to be produced then nothing will be returned. Below is an example showing how the features can be accessed and printed out.

```
ann = seqann.annotate(sequence, "HLA-A")
for feat in ann.annotation:
    print(feat, ann.gfe, str(ann.annotation[feat].seq), sep="\t")
```

CHAPTER 6

Dependencies

- Clustal Omega 1.2.0 or higher
- Python 3.6
- blastn

CHAPTER 7

Installation

7.1 Stable release

To install SeqAnn, run this command in your terminal:

```
$ pip install seqann
```

This is the preferred method to install SeqAnn, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

7.2 From sources

The sources for SeqAnn can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/nmdp-bioinformatics/seqann
```

Or download the tarball:

```
$ curl  -OL https://github.com/nmdp-bioinformatics/seqann/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 8

Annotation Algorithm

Note: There are several places where hard-coded logic was added to make the algorithm work with certain sequences. Hard coded logic is marked by **HARD CODED LOGIC** in the code. For instance, in *seqann.seq_search* I added logic to annotate exon 8 for class I last, because mapping it first causes issues due to the size of the feature.

1. **Check if locus is provided** (*seqann.blast_cmd*)
    - yes, then continue to step 2
    - no, then blast to get locus
2. **Check if any exact matches exist** (*Reference Data*)
    - yes, then return annotation associated with the exact match and go to step 7
    - no, then go on to step 3
3. **Blast sequence and get list of alleles** (*seqann.blast_cmd*)
    - If all of the returned sequences are partial then the last sequence will be replaced with a fully characterized sequence.
4. **Iterate through the list and try to annotate with reference sequences** (*seqann.seq_search*)
    - Break reference up into features
    - Search for each feature in the provided sequence. When a feature is found record the coordinates and remove the mapped sequence from the unknown coordinates.
    - If all features are mapped, then go to step 7, else..
    - Try and assemble the remaining features based on what has already been mapped. Since we know the coordinates of the mapped features and the remaining unmapped sequence, we can determine if the unmapped sequences fall between two mapped features or at the ends/beginning after/before mapped sequences.
    - If all features could be mapped, then go to step 7 else go back to step 4A using any partial annotations for each reference sequence. If no annotation could be created after searching all of the reference sequences, then move on to step 5.

- Partial annotations are *Annotation* objects with `mapping`, `blocks`, `covered` attributes. The `mapping` attribute is a dictionary with each position being a key and the values being features if they are mapped. The `blocks` attribute is a list of lists, which each list representing the positions of the unmapped parts of the sequence. If the whole sequence isn't mapped then there will only be one list (block).

5. **Loop through each reference sequence and do targeted alignments** (*seqann.align*)
   - Break up each reference sequence into features and create feature combos that will be used for doing alignments. Order the feature combos by the ones that make the most sense first.
   - Do targeted alignments on all of the remaining blocks of sequences that have not yet been mapped.
     - If a high enough proportion of the unmapped sequence maps and the deletion/insertion rate is low enough, then extract the unmapped sequence from the alignment and map it.
     - If all features are mapped then go to step 7, else..
     - run step 4 with the updated partial annotation to see if the annotation can now be assembled. Go to step 7 if all features are mapped else..
   - Loop through all feature combinations for all reference sequences. This slows down the annotation if it's very novel. For instance, if it's a new feature sequence and that specific feature has only been reported in IMGT/HLA a few time for a given locus. The acceptance rate for the alignments is decreased slightly after each loop. For class I that decrease stops after the second reference sequence, but for class II it will keep going lower.
   - Rerun targeted alignment but with exons only combinations.
6. **Do a full sequence alignment and use any partial annotation** (*seqann.align*)
   - If this fails and the rerun flag is set to `True`, then rerun the whole annotation process starting from step 1. This time, skip the first reference allele that was used for doing the annotation and increase the number of reference alleles used by 1.
7. **Generate GFE notation** (*seqann.gfe*)
   - Once a complete annotation is generated the GFE notation will be made
   - If the sequence only contains A,T,C or G, then a GFE notation can be created

CHAPTER 9

BioSQL Database

9.1 Running Container

To get the IPD-IMGT/HLA BioSQL Docker image, run this command in your terminal:

```
$ docker pull nmdpbioinformatics/imgt_biosqldb
```

You can then run the datbase with the following command in your terminal:

```
$ docker run -d --name imgt_biosqldb -p 3306:3306 \
    -e MYSQL_ROOT_PASSWORD=my-secret-pw nmdpbioinformatics/imgt_biosqldb
```

If you have a mysql database running locally already, then you can change the first number in the port mapping to something else. If you change the port then remember to export the `BIOSQLPORT` environment variable to whatever you used. A password is required and the `seqann` package can access your password if you set the `BIOSQLPASS` environment variable.

9.2 Building Locally

If you want certain IPD-IMGT/HLA database verions that aren't loaded into the publicly available docker image, then you can build the image locally with the database versions you want. The sources for IPD-IMGT/HLA BioSQL database can be downloaded from the **'Github repo'_**.

First clone the public repository:

```
$ git clone git://github.com/nmdp-bioinformatics/imgt_biosqldb
```

Then build the docker image and provide the `RELEASES` you want to use as an argument.

```
$ docker build -t imgt_biosqldb:3240-3250 --build-arg RELEASES="3240,3250" .
```

Once the image has finished building, you can run the database as described above.

CHAPTER 10

Creating blastn files

Note: Make sure blastn is properly installed before running!

1) Download the allele list and the `_gen` and `_nuc` fasta files for each locus
2) Create the blast files by running the **ngs-imgt-db** perl script

```
$ ngs-imgt-db -i /path/to/imgt/files -o /output/dir
```

3) Add the new blast files to the seqann/data/blast directory and check them in.

CHAPTER 11

Debugging

In order to debug your code you must first create a logging instance in your script.

```
import logging
logging.basicConfig(format='%(asctime)s - %(name)-35s - %(levelname)-5s - %(message)s
↪',
            datefmt='%m/%d/%Y %I:%M:%S %p',
            level=logging.INFO)
```

Once you have the logging set up, you can then pass a dictionary to the debug parameter. The dictionary should have keys representing a main process and values for how verbose the output should be for those proecesses. If a process is not present in the debug dictionary, then no logging will be generated for that process.

```
seqann = BioSeqAnn(debug={"seqann": 5, "align":1, "refdata":0, "seq_search": 5, "gfe
↪": 4})
```

Instead of using the debug parameter, you can use the verbose and verbosity parameters to see the same level logging for each process. Use these parameters when you want to see the most logging possible for each process.

```
seqann = BioSeqAnn(verbose=True, verbosity=5)
```

CHAPTER 12

Testing

Warning: Before running tests clustalo, blastn and all the required python packages must be installed!

To run all test simply execute the following command in the top directory of the SeqAnn repository.

```
$ python -m unittest tests
```

12.1 Different tests

Note: If you don't have a imgt_biosql db running then not all of the test will run!

- test_seqann
- test_align
- test_blast
- test_feature
- test_gfe
- test_refdata
- test_seqsearch
- test_util

12.2 Running specific tests

You can test a specific test by providing the full test path on the command line.

```
$ python -m unittest tests.test_seqann.TestBioSeqAnn.test_004_ambig
```

CHAPTER 13

seqann package

13.1 seqann.sequence_annotation

class seqann.sequence_annotation.**BioSeqAnn**(*server: <module 'BioSQL.BioSeqDatabase' from '/home/docs/checkouts/readthedocs.org/user_builds/seqann/envs/stable/lib/python3.6/site-packages/BioSQL/BioSeqDatabase.py'> = None*, *dbversion: str = '3310'*, *datfile: str = ''*, *verbose: bool = False*, *verbosity: int = 0*, *pid: str = 'NA'*, *kir: bool = False*, *align: bool = False*, *load_features: bool = False*, *store_features: bool = False*, *refdata: seqann.models.reference_data.ReferenceData = None*, *cached_features: Dict[KT, VT] = None*, *safemode: bool = False*, *debug: Dict[KT, VT] = None*)

Bases: seqann.models.base_model_.Model

```
from seqann import BioSeqAnn
seqann1 = BioSeqAnn()
seqann2 = BioSeqAnn(dbversion="3300", verbose=True, verbosity=3)
seqann3 = BioSeqAnn(debug={"align":4}, safe)
```

Parameters

- **server** (*BioSQL Database*) – A BioSQL database to use for retriving the sequence features. Using a BioSQL DB will speed up the annotations dramatically.
- **dbversion** (str) – The IPD-IMGT/HLA or KIR database release.
- **datfile** (str) – The IPD-IMGT/HLA or KIR dat file to use in place of the server parameter.
- **pid** (str) – A process label that can be provided to help track the logging output.

- **load_features** (bool) – Flag for downloading all gene features and accessions from the feature service.
- **store_features** (bool) – Flag for caching all features and their corresponding accessions.
- **cached_features** (dict) – Dictionary containing all the features from the feature service.
- **kir** (bool) – Flag for indicating the input sequences are from the KIR gene system.
- **align** (bool) – Flag for producing the alignments along with the annotations.
- **verbose** (bool) – Flag for running in verbose mode.
- **verbosity** (int) – Numerical value to indicate how verbose the output will be in verbose mode.
- **debug** (dict) – Dictionary containing names of steps that you want to debug.
- **safemode** (bool) – Flag for running the annotations in safemode. No alignments will be done if no feature matches were made. This can prevent the alignment step for running for too long on bad sequences.

annotate (*sequence: Bio.Seq.Seq = None*, *locus: str = None*, *nseqs: int = 20*, *alignseqs: int = 10*, *skip: List[T] = []*, *rerun: bool = True*, *full: bool = True*) → seqann.models.annotation.Annotation

annotate - method for annotating a BioPython sequence

Parameters

- **sequence** (*Seq*) – The input consensus sequence.
- **locus** (str) – The gene locus associated with the sequence.
- **nseqs** (int) – The number of blast sequences to use.
- **alignseqs** (int) – The number of sequences to use for targeted alignments.
- **skip** (List) – A list of alleles to skip for using as a reference. This is used for validation and testing.

Return type *Annotation*

Returns: The annotate function return an Annotation object that contains the sequence features and names associated with them.

Example output:

```
{
     'complete_annotation': True,
     'annotation': {'exon_1': SeqRecord(seq=Seq('AGAGACTCTCCCG',␣
↪SingleLetterAlphabet()), id='HLA:HLA00630', name='HLA:HLA00630',␣
↪description='HLA:HLA00630 DQB1*03:04:01 597 bp', dbxrefs=[]),
                    'exon_2': SeqRecord(seq=Seq(
↪'AGGATTTCGTGTACCAGTTTAAGGCCATGTGCTACTTCACCAACGGGACGGAGC...GAG',␣
↪SingleLetterAlphabet()), id='HLA:HLA00630', name='HLA:HLA00630',␣
↪description='HLA:HLA00630 DQB1*03:04:01 597 bp', dbxrefs=[]),
                    'exon_3': SeqRecord(seq=Seq(
↪'TGGAGCCCACAGTGACCATCTCCCCATCCAGGACAGAGGCCCTCAACCACCACA...ATG',␣
↪SingleLetterAlphabet()), id='HLA:HLA00630', name='<unknown name>',␣
↪description='HLA:HLA00630', dbxrefs=[])},
     'features': {'exon_1': SeqFeature(FeatureLocation(ExactPosition(0),␣
↪ExactPosition(13), strand=1), type='exon_1'),
```

(continues on next page)

(continued from previous page)

```
                    'exon_2': SeqFeature(FeatureLocation(ExactPosition(13),␣
↪ExactPosition(283), strand=1), type='exon_2')
                    'exon_3': SeqFeature(FeatureLocation(ExactPosition(283),
↪ ExactPosition(503), strand=1), type='exon_3')},
     'method': 'nt_search and clustalo',
     'gfe': 'HLA-Aw2-1-1-1-1-1-1-1-1-1-1-1-1-1-1-4',
     'seq': SeqRecord(seq=Seq(
↪'AGAGACTCTCCCGAGGATTTCGTGTACCAGTTTAAGGCCATGTGCTACTTCACC...ATG',␣
↪SingleLetterAlphabet()), id='HLA:HLA00630', name='HLA:HLA00630',␣
↪description='HLA:HLA00630 DQB1*03:04:01 597 bp', dbxrefs=[])
}
```

Example usage:

```
>>> from Bio.Seq import Seq
>>> from seqann import BioSeqAnn
>>> sequence = Seq('AGAGACTCTCCCGAGGATTTCGTGTACCAGTTTAAGGCCATGTGCTACTTCACC')
>>> seqann = BioSeqAnn()
>>> ann = seqann.annotate(sequence)
>>> for f in ann.annotation:
...     print(f, ann.method, str(ann.annotation[f].seq), sep="       ")
exon_2  nt_search and clustalo ␣
↪AGGATTTCGTGTACCAGTTTAAGGCCATGTGCTACTTCACCAACGGGACGGAGCGCGTGCGTTATGTGACCAGATACATCTATAACCGAG
exon_3  nt_search and clustalo ␣
↪TGGAGCCCACAGTGACCATCTCCCCATCCAGGACAGAGGCCCTCAACCACCACAACCTGCTGGTCTGCTCAGTGACAGATTTCTATCCAG
exon_1  nt_search and clustalo  AGAGACTCTCCCG
exon_4  nt_search and clustalo  GGGCTCAGTCTGAATCTGCCCAGAGCAAGATG
```

ref_align (*found_seqs*, *sequence: Bio.Seq.Seq = None*, *locus: str = None*, *annotation: seqann.models.annotation.Annotation = None*, *partial_ann: seqann.models.annotation.Annotation = None*, *run: int = 0*, *cutoff: float = 0.9*) → seqann.models.annotation.Annotation

ref_align - Method for doing targeted alignments on partial annotations

Parameters

- **found_seqs** (*Seq*) – The input sequence record.
- **sequence** (*Seq*) – The input sequence record.
- **locus** (str) – The gene locus associated with the sequence.
- **annotation** (*Annotation*) – The incomplete annotation from a previous iteration.
- **partial_ann** (*Annotation*) – The partial annotation after looping through all of the blast sequences.

Return type *Annotation*

add_alignment (*ref_seq*, *annotation*) → seqann.models.annotation.Annotation

add_alignment - method for adding the alignment to an annotation

Parameters

- **ref_seq** (*List*) – List of reference sequences
- **annotation** (Annotation) – The complete annotation

Return type *Annotation*

seqann.sequence_annotation.**getblocks** (*coords*)

13.2 seqann.seq_search

class seqann.seq_search.**SeqSearch**(*verbose: bool = False*, *verbosity: int = 0*)

Bases: seqann.models.base_model_.Model

This is a class for annotating a BioPython sequence without using alignment

Parameters

- **verbose** (bool) – Flag for running in verbose mode.
- **verbosity** (int) – Numerical value to indicate how verbose the output will be in verbose mode.

Example usage:

```
>>> from seqann.seq_search import SeqSearch
>>> seqsrch = SeqSearch()
```

classmethod from_dict(*dikt*) → seqann.seq_search.SeqSearch

Returns the dict as a model

Parameters **dikt** – A dict.

Type dict

Returns The SeqSearch of this SeqSearch.

Return type *SeqSearch*

search_seqs(*seqrec*, *in_seq*, *locus*, *run=0*, *partial_ann=None*)

search_seqs - method for annotating a BioPython sequence without alignment

Parameters

- **seqrec** (*SeqRecord*) – The reference sequence
- **locus** (*str*) – The gene locus associated with the sequence.
- **in_seq** (*SeqRecord*) – The input sequence
- **run** (*int*) – The number of runs that have been done
- **partial_ann** (*Annotation*) – A partial annotation from a previous step

Return type *Annotation*

Example usage:

```
>>> from Bio.Seq import Seq
>>> from seqann.seq_search import SeqSearch
>>> inseq = Seq('AGAGACTCTCCCGAGGATTTCGTGTACCAGTTTAAGGCCATGTGCTACTTCACC')
>>> sqsrch = SeqSearch()
>>> ann = sqsrch.search_seqs(refseqs, inseq)
```

verbose

Gets the verbose of this SeqSearch.

Returns The verbose of this SeqSearch.

Return type bool

verbosity

Gets the verbosity of this SeqSearch.

Returns The verbosity of this SeqSearch.

Return type int

seqann.seq_search.**loctype**(*s1*, *e1*, *s2*, *e2*)

seqann.seq_search.**getblocks**(*coords*)

13.3 seqann.gfe

class seqann.gfe.**GFE**(*url='http://feature.nmdp-bioinformatics.org', loci=['KIR2DP1', 'KIR2DL5A', 'KIR2DS4', 'HLA-DRA', 'HLA-DPA1', 'HLA-DQA1', 'HLA-DPB1', 'KIR2DS2', 'KIR3DP1', 'HLA-DRB4', 'KIR2DL1', 'KIR2DS5', 'HLA-DRB3', 'KIR2DS3', 'KIR3DL1', 'HLA-A', 'HLA-DRB5', 'KIR2DL4', 'HLA-DQB1', 'KIR3DL2', 'HLA-B', 'KIR3DS1', 'KIR2DL5B', 'HLA-DRB1', 'KIR3DL3', 'KIR2DS1', 'HLA-C'], load_features=False, store_features=False, cached_features=None, verbose=False, pid='NA', verbosity=0*)

Bases: object

This class is used for converting annotations into GFE notations.

Example:

```
>>> from Bio import SeqIO
>>> from BioSQL import BioSeqDatabase
>>> from seqann.sequence_annotation import BioSeqAnn
>>> from pygfe.pygfe import pyGFE
>>> seq_file = 'test_dq.fasta'
>>> gfe = pyGFE()
>>> server = BioSeqDatabase.open_database(driver="pymysql", user="root",
...                                       passwd="", host="localhost",
...                                      db="bioseqdb")
>>> seqann = BioSeqAnn(server=server)
>>> seq_rec = list(SeqIO.parse(seq_file, 'fasta'))[0]
>>> annotation = seqann.annotate(seq_rec, "HLA-DQB1")
>>> features, gfe = gfe.get_gfe(annotation, "HLA-DQB1")
>>> print(gfe)
HLA-DQB1w0-4-0-141-0-12-0-4-0-0-0-0-0
```

load_features()
Loads all the known features from the feature service

locus_features(*locus*)
Returns all features associated with a locus

Parameters **locus** (str) – string containing HLA locus.

Return type dict

get_gfe(*annotation*, *locus*)
creates GFE from a sequence annotation

Parameters

- **locus** (str) – The gene locus
- **annotation** (List) – An sequence annotation object

Return type List

Returns: The GFE notation and the associated features in an array

13.4 seqann.blast_cmd

seqann.blast_cmd.**has_hla**(*x*)

seqann.blast_cmd.**blastn**(*sequences*, *locus*, *nseqs*, *kir=False*, *verbose=False*, *refdata=None*, *evalue=10*)

Gets the a list of alleles that are the most similar to the input sequence

Parameters

- **sequences** (*SeqRecord*) – The input sequence record.
- **locus** (str) – The gene locus associated with the sequence.
- **nseqs** (int) – The incomplete annotation from a previous iteration.
- **evalue** (int) – The evalue to use (default = 10)
- **kir** (bool) – Run with KIR or not
- **verbose** (bool) – Run in versboe
- **refdata** (*Reference Data*) – An object with reference data

Return type *Blast*

Example usage:

```
>>> from Bio.Seq import Seq
>>> from seqann.blast_cmd import blastn
>>> sequence = Seq('AGAGACTCTCCCGAGGATTTCGTGTACCAGTTTAAGGCCATGTGCTACTTCACC')
>>> blast = blastn(sequence, locus, nseqs)
```

seqann.blast_cmd.**get_locus**(*sequences*, *kir=False*, *verbose=False*, *refdata=None*, *evalue=10*)

Gets the locus of the sequence by running blastn

Parameters

- **sequences** – sequenences to blast
- **kir** – bool whether the sequences are KIR or not

Return type str

Example usage:

```
>>> from Bio.Seq import Seq
>>> from seqann.blast_cmd import get_locus
>>> sequence = Seq('AGAGACTCTCCCGAGGATTTCGTGTACCAGTTTAAGGCCATGTGCTACTTCACC')
>>> locus = get_locus(sequence)
```

13.5 seqann.align

seqann.align.**flatten**(*l*)

seqann.align.**align_seqs**(*found_seqs*, *sequence*, *locus*, *start_pos*, *missing*, *annotated*, *cutoff=0.9*, *verbose=False*, *verbosity=0*)

align_seqs - Aligns sequences with clustalo

Parameters

- **found_seqs** (List) – List of the reference sequences
- **sequence** (*SeqRecord*) – The input consensus sequence.
- **locus** (str) – The gene locus associated with the sequence.
- **annotated** (dict) – dictonary of the annotated features
- **start_pos** (int) – Where the reference sequence starts
- **missing** (List) – List of the unmapped features
- **cutoff** (float) – The alignment cutoff
- **verbose** (bool) – Flag for running in verbose mode.
- **verbosity** (int) – Numerical value to indicate how verbose the output will be in verbose mode.

Return type *Annotation*

seqann.align.**find_features**(*feats*, *sequ*, *annotated*, *start_pos*, *cutoff*)

find_features - Finds the reference sequence features in the alignments and records the positions

Parameters

- **feats** (dict) – Dictonary of sequence features
- **sequ** (List) – The sequence alignment for the input sequence
- **annotated** (dict) – dictonary of the annotated features
- **start_pos** (int) – Where the reference sequence starts
- **missing** (List) – List of the unmapped features
- **cutoff** (float) – The alignment cutoff
- **verbose** (bool) – Flag for running in verbose mode.
- **verbosity** (int) – Numerical value to indicate how verbose the output will be in verbose mode.

Return type List

seqann.align.**resolve_feats**(*feat_list*, *seqin*, *seqref*, *start*, *locus*, *missing*, *verbose=False*, *verbosity=0*)

resolve_feats - Resolves features from alignments

Parameters

- **feat_list** (List) – List of the found features
- **seqin** (str) – The input sequence
- **locus** (str) – The input locus
- **start** (int) – Where the sequence start in the alignment
- **missing** (List) – List of the unmapped features
- **verbose** (bool) – Flag for running in verbose mode.
- **verbosity** (int) – Numerical value to indicate how verbose the output will be in verbose mode.

Return type *Annotation*

seqann.align.**count_diffs**(*align*, *feats*, *inseq*, *locus*, *cutoff*, *verbose=False*, *verbosity=0*)

count_diffs - Counts the number of mismatches, gaps, and insertions and then determines if those are within an acceptable range.

Parameters

- **align** (List) – The alignment
- **feats** (dict) – Dictonary of the features
- **locus** (str) – The gene locus associated with the sequence.
- **inseq** (str) – The input sequence
- **cutoff** (float) – The alignment cutoff
- **verbose** (bool) – Flag for running in verbose mode.
- **verbosity** (int) – Numerical value to indicate how verbose the output will be in verbose mode.

Return type List

CHAPTER 14

seqann models

14.1 Annotation

class seqann.models.annotation.**Annotation**(*missing: Dict[KT, VT] = None*, *ambig: Dict[KT, VT] = None*, *seq: Bio.SeqRecord.SeqRecord = None*, *features: Dict[KT, VT] = None*, *covered: int = None*, *annotation: Dict[KT, VT] = {}*, *blocks: List[List[int]] = None*, *method: str = None*, *mapping: Dict[KT, VT] = None*, *refmissing: List[str] = None*, *exact_match: List[str] = None*, *exact: bool = False*, *structure: List[seqann.feature_client.models.feature.Feature] = None*, *complete_annotation: bool = False*, *gfe: str = None*)

Bases: seqann.models.base_model_.Model

classdocs

classmethod from_dict(*dikt*) → seqann.models.annotation.Annotation

Returns the dict as a model

Parameters **dikt** – A dict.

Type dict

Returns The Annotation of this Annotation.

Return type *Annotation*

complete_annotation

Gets the complete_annotation of this Annotation.

Returns The complete_annotation of this Annotation.

Return type bool

exact
Gets the exact of this Annotation.

Returns The exact of this Annotation.

Return type bool

features
Gets the features of this Annotation.

Returns The features of this Annotation.

Return type Dict

structure
Gets the structure of this Annotation.

Returns The structure of this Annotation.

Return type List[*Feature*]

covered
Gets the coverage of this Annotation.

Returns The covered of this Annotation.

Return type int

seq
Gets the coverage of this Annotation.

Returns The seq of this Annotation.

Return type SeqRecord

ambig
Gets the ambig of this Annotation.

Returns The ambig of this Annotation.

Return type Dict

method
Gets the method of this Annotation.

Returns The method of this Annotation.

Return type str

missing
Gets the missing of this Annotation.

Returns The missing of this Annotation.

Return type Dict

mapping
Gets the coverage of this Annotation.

Returns The seq of this Annotation.

Return type Dict

refmissing
Gets the refmissing of this Annotation.

Returns The refmissing of this Annotation.

Return type List[str]

exact_match
Gets the exact_match of this Annotation.

Returns The exact_match of this Annotation.

Return type List[str]

annotation
Gets the coverage of this Annotation.

Returns The seq of this Annotation.

Return type Dict

blocks
Gets the coverage of this Annotation.

Returns The blocks of this Annotation.

Return type List[List[int]]

gfe
Gets the coverage of this Annotation.

Returns The gfe of this Annotation.

Return type Dict

aligned
Gets the coverage of this Annotation.

Returns The aligned of this Annotation.

Return type Dict

check_annotation()

clean()

14.2 Reference Data

seqann.models.reference_data.**download_dat**(*url*, *dat*)

class seqann.models.reference_data.**ReferenceData**(*server: <module 'BioSQL.BioSeqDatabase' from '/home/docs/checkouts/readthedocs.org/user_builds/seqann/en packages/BioSQL/BioSeqDatabase.py'> = None*, *datafile: str = None*, *dbversion: str = '3310'*, *alleles: List[T] = None*, *seqdata: Dict[KT, VT] = None*, *hladata: Dict[KT, VT] = None*, *featuredata=None*, *kir: bool = False*, *alignments: bool = False*, *verbose: bool = False*, *verbosity: int = 0*)

Bases: seqann.models.base_model_.Model

classdocs

classmethod from_dict(*dikt*) → seqann.models.reference_data.ReferenceData
Returns the dict as a model

Parameters **dikt** – A dict.

Type dict

Returns The ReferenceData of this ReferenceData.

Return type *ReferenceData*

server
Gets the server of this ReferenceData.

Returns The server of this ReferenceData.

Return type BioSeqDatabase

verbose
Gets the server of this ReferenceData.

Returns The server of this ReferenceData.

Return type BioSeqDatabase

verbosity
Gets the server of this ReferenceData.

Returns The server of this ReferenceData.

Return type BioSeqDatabase

alignments
Gets the alignments of this ReferenceData.

Returns The alignments of this ReferenceData.

Return type BioSeqDatabase

datafile
Gets the datafile of this ReferenceData.

Returns The datafile of this ReferenceData.

Return type str

dbversion
Gets the dbversion of this ReferenceData.

Returns The dbversion of this ReferenceData.

Return type str

structures
Gets the structures of this ReferenceData.

Returns The structures of this ReferenceData.

Return type Dict

structure_max
Gets the structure_max of this ReferenceData.

Returns The structure_max of this ReferenceData.

Return type Dict

hlaref
Gets the hlaref of this ReferenceData.

Returns The hlaref of this ReferenceData.

Return type Dict

seqref
Gets the seqref of this ReferenceData.

Returns The seqref of this ReferenceData.

Return type Dict

blastdb
Gets the blastdb of this ReferenceData.

Returns The blastdb of this ReferenceData.

Return type str

struct_order
Gets the struct_order of this ReferenceData.

Returns The struct_order of this ReferenceData.

Return type Dict

feature_lengths
Gets the feature_lengths of this ReferenceData.

Returns The feature_lengths of this ReferenceData.

Return type Dict

hla_names
Gets the hla_names of this ReferenceData.

Returns The hla_names of this ReferenceData.

Return type Dict

kir
Gets the kir of this ReferenceData.

Returns The kir of this ReferenceData.

Return type bool

hla_loci
Gets the hla_loci of this ReferenceData.

Returns The hla_loci of this ReferenceData.

Return type List[str]

server_avail
Gets the server_avail of this ReferenceData.

Returns The server_avail of this ReferenceData.

Return type bool

search_refdata(*seq*, *locus*)
This checks to see if a sequence already exists in the reference data. If it does, then it'll return the known annotation.

Returns The Annotation of associated with the input sequence

Return type *Annotation*

Example:

```
>>> from Bio.Seq import Seq
>>> from seqann.models.reference_data import ReferenceData
>>> sequence = Seq('AGAGACTCTCCCGAGGATTTCGTGTACCAGTTTAAGGCCATGTGCTACTTCACC')
>>> refdata = ReferenceData()
>>> matched_annotation = refdata.search_refdata(sequence, locus)
```

seqrecord(*allele*, *locus*)
Gets the Annotation from the found sequence

Returns The Annotation from the found sequence

Return type *Annotation*

seqannotation(*seqrecord*, *allele*, *loc*)
Gets the Annotation from the found sequence

Returns The Annotation from the found sequence

Return type *Annotation*

14.3 Blast

class seqann.models.blast.**Blast**(*failed: bool = None*, *match_seqs: List[BioSQL.BioSeq.DBSeqRecord] = None*, *alleles: List[str] = None*)
Bases: seqann.models.base_model_.Model

classdocs

classmethod from_dict(*dikt*) → seqann.models.blast.Blast
Returns the dict as a model

Parameters **dikt** – A dict.

Type dict

Returns The Blast of this Blast.

Return type *Blast*

match_seqs
Gets the match_seqs of this Blast.

Returns The match_seqs of this Blast.

Return type List[str]

alleles
Gets the alleles of this Blast.

Returns The alleles of this Blast.

Return type List[str]

failed
Gets the failed of this Blast.

Returns The failed of this Blast.

Return type bool

CHAPTER 15

seqann.feature_client package

15.1 Subpackages

15.1.1 seqann.feature_client.apis package

Submodules

seqann.feature_client.apis.features_api module

No descripton provided (generated by Swagger Codegen https://github.com/swagger-api/swagger-codegen)

OpenAPI spec version: 1.0

Generated by: https://github.com/swagger-api/swagger-codegen.git

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

class seqann.feature_client.apis.features_api.**FeaturesApi**(*api_client=None*)
Bases: object

NOTE: This class is auto generated by the swagger code generator program. Do not edit the class manually. Ref: https://github.com/swagger-api/swagger-codegen

create_feature(***kwargs*)
Create an enumerated sequence feature

This method makes a synchronous HTTP request by default. To make an asynchronous HTTP request, please define a *callback* function to be invoked when receiving the re-

sponse. >>> def callback_function(response): >>> pprint(response) >>> >>> thread = api.create_feature(callback=callback_function)

Parameters

- **function** (*callback*) – The callback function for asynchronous request. (optional)
- **body** (FeatureRequest) –

Returns Feature If the method is called asynchronously, returns the request thread.

create_feature_with_http_info(***kwargs*)
Create an enumerated sequence feature

This method makes a synchronous HTTP request by default. To make an asynchronous HTTP request, please define a *callback* function to be invoked when receiving the response. >>> def callback_function(response): >>> pprint(response) >>> >>> thread = api.create_feature_with_http_info(callback=callback_function)

Parameters

- **function** (*callback*) – The callback function for asynchronous request. (optional)
- **body** (FeatureRequest) –

Returns Feature If the method is called asynchronously, returns the request thread.

get_feature_by_path(*locus*, *term*, *rank*, *accession*, ***kwargs*)
Retrieve an enumerated sequence feature

This method makes a synchronous HTTP request by default. To make an asynchronous HTTP request, please define a *callback* function to be invoked when receiving the response. >>> def callback_function(response): >>> pprint(response) >>> >>> thread = api.get_feature_by_path(locus, term, rank, accession, callback=callback_function)

Parameters

- **function** (*callback*) – The callback function for asynchronous request. (optional)
- **locus** (*str*) – locus name or URI (required)
- **term** (*str*) – Sequence Ontology (SO) term name, accession, or URI (required)
- **rank** (*int*) – feature rank, must be at least 1 (required)
- **accession** (*int*) – accession, must be at least 1 (required)

Returns Feature If the method is called asynchronously, returns the request thread.

get_feature_by_path_with_http_info(*locus*, *term*, *rank*, *accession*, ***kwargs*)
Retrieve an enumerated sequence feature

This method makes a synchronous HTTP request by default. To make an asynchronous HTTP request, please define a *callback* function to be invoked when receiving the response. >>> def callback_function(response): >>> pprint(response) >>> >>> thread = api.get_feature_by_path_with_http_info(locus, term, rank, accession, callback=callback_function)

Parameters

- **function** (*callback*) – The callback function for asynchronous request. (optional)
- **locus** (*str*) – locus name or URI (required)
- **term** (*str*) – Sequence Ontology (SO) term name, accession, or URI (required)
- **rank** (*int*) – feature rank, must be at least 1 (required)

- **accession** (*int*) – accession, must be at least 1 (required)

Returns Feature If the method is called asynchronously, returns the request thread.

get_feature_by_query(***kwargs*)
Retrieve an enumerated sequence feature

This method makes a synchronous HTTP request by default. To make an asynchronous HTTP request, please define a *callback* function to be invoked when receiving the response. >>> def callback_function(response): >>> pprint(response) >>> >>> thread = api.get_feature_by_query(callback=callback_function)

Parameters

- **function** (*callback*) – The callback function for asynchronous request. (optional)
- **locus** (*str*) – locus name or URI
- **term** (*str*) – Sequence Ontology (SO) term name, accession, or URI
- **rank** (*int*) – feature rank, must be at least 1
- **accession** (*int*) – accession, must be at least 1

Returns Feature If the method is called asynchronously, returns the request thread.

get_feature_by_query_with_http_info(***kwargs*)
Retrieve an enumerated sequence feature

This method makes a synchronous HTTP request by default. To make an asynchronous HTTP request, please define a *callback* function to be invoked when receiving the response. >>> def callback_function(response): >>> pprint(response) >>> >>> thread = api.get_feature_by_query_with_http_info(callback=callback_function)

Parameters

- **function** (*callback*) – The callback function for asynchronous request. (optional)
- **locus** (*str*) – locus name or URI
- **term** (*str*) – Sequence Ontology (SO) term name, accession, or URI
- **rank** (*int*) – feature rank, must be at least 1
- **accession** (*int*) – accession, must be at least 1

Returns Feature If the method is called asynchronously, returns the request thread.

list_features(*locus*, ***kwargs*)
List the enumerated sequence features at a locus

This method makes a synchronous HTTP request by default. To make an asynchronous HTTP request, please define a *callback* function to be invoked when receiving the response. >>> def callback_function(response): >>> pprint(response) >>> >>> thread = api.list_features(locus, callback=callback_function)

Parameters

- **function** (*callback*) – The callback function for asynchronous request. (optional)
- **locus** (*str*) – locus name or URI (required)

Returns list[Feature] If the method is called asynchronously, returns the request thread.

list_features_with_http_info(*locus*, ***kwargs*)
List the enumerated sequence features at a locus

This method makes a synchronous HTTP request by default. To make an asynchronous HTTP request, please define a *callback* function to be invoked when receiving the response. >>> def callback_function(response): >>> pprint(response) >>> >>> thread = api.list_features_with_http_info(locus, callback=callback_function)

Parameters

- **function** (*callback*) – The callback function for asynchronous request. (optional)
- **locus** (*str*) – locus name or URI (required)

Returns list[Feature] If the method is called asynchronously, returns the request thread.

list_features_0(*locus*, *term*, ***kwargs*)
List the enumerated sequence features matching a term at a locus

This method makes a synchronous HTTP request by default. To make an asynchronous HTTP request, please define a *callback* function to be invoked when receiving the response. >>> def callback_function(response): >>> pprint(response) >>> >>> thread = api.list_features_0(locus, term, callback=callback_function)

Parameters

- **function** (*callback*) – The callback function for asynchronous request. (optional)
- **locus** (*str*) – locus name or URI (required)
- **term** (*str*) – Sequence Ontology (SO) term name, accession, or URI (required)

Returns list[Feature] If the method is called asynchronously, returns the request thread.

list_features_0_with_http_info(*locus*, *term*, ***kwargs*)
List the enumerated sequence features matching a term at a locus

This method makes a synchronous HTTP request by default. To make an asynchronous HTTP request, please define a *callback* function to be invoked when receiving the response. >>> def callback_function(response): >>> pprint(response) >>> >>> thread = api.list_features_0_with_http_info(locus, term, callback=callback_function)

Parameters

- **function** (*callback*) – The callback function for asynchronous request. (optional)
- **locus** (*str*) – locus name or URI (required)
- **term** (*str*) – Sequence Ontology (SO) term name, accession, or URI (required)

Returns list[Feature] If the method is called asynchronously, returns the request thread.

list_features_1(*locus*, *term*, *rank*, ***kwargs*)
List the enumerated sequence features matching a term and rank at a locus

This method makes a synchronous HTTP request by default. To make an asynchronous HTTP request, please define a *callback* function to be invoked when receiving the response. >>> def callback_function(response): >>> pprint(response) >>> >>> thread = api.list_features_1(locus, term, rank, callback=callback_function)

Parameters

- **function** (*callback*) – The callback function for asynchronous request. (optional)
- **locus** (*str*) – locus name or URI (required)
- **term** (*str*) – Sequence Ontology (SO) term name, accession, or URI (required)
- **rank** (*int*) – feature rank, must be at least 1 (required)

Returns list[Feature] If the method is called asynchronously, returns the request thread.

list_features_1_with_http_info(*locus*, *term*, *rank*, ***kwargs*)
List the enumerated sequence features matching a term and rank at a locus

This method makes a synchronous HTTP request by default. To make an asynchronous HTTP request, please define a *callback* function to be invoked when receiving the response. >>> def callback_function(response): >>> pprint(response) >>> >>> thread = api.list_features_1_with_http_info(locus, term, rank, callback=callback_function)

Parameters

- **function** (*callback*) – The callback function for asynchronous request. (optional)
- **locus** (*str*) – locus name or URI (required)
- **term** (*str*) – Sequence Ontology (SO) term name, accession, or URI (required)
- **rank** (*int*) – feature rank, must be at least 1 (required)

Returns list[Feature] If the method is called asynchronously, returns the request thread.

Module contents

15.1.2 seqann.feature_client.models package

Submodules

seqann.feature_client.models.feature module

No descripton provided (generated by Swagger Codegen https://github.com/swagger-api/swagger-codegen)

OpenAPI spec version: 1.0

Generated by: https://github.com/swagger-api/swagger-codegen.git

class seqann.feature_client.models.feature.**Feature**(*locus=None*, *term=None*, *rank=None*, *accession=None*, *sequence=None*, *hash_code=None*)

Bases: object

NOTE: This class is auto generated by the swagger code generator program. Do not edit the class manually.

locus
Gets the locus of this Feature.

Returns The locus of this Feature.

Return type str

term
Gets the term of this Feature.

Returns The term of this Feature.

Return type str

rank
Gets the rank of this Feature.

Returns The rank of this Feature.

Return type int

accession
Gets the accession of this Feature.

Returns The accession of this Feature.

Return type int

sequence
Gets the sequence of this Feature.

Returns The sequence of this Feature.

Return type str

hash_code
Gets the hash_code of this Feature.

Returns The hash_code of this Feature.

Return type int

to_dict()
Returns the model properties as a dict

to_str()
Returns the string representation of the model

seqann.feature_client.models.feature_request module

No descripton provided (generated by Swagger Codegen https://github.com/swagger-api/swagger-codegen)

OpenAPI spec version: 1.0

Generated by: https://github.com/swagger-api/swagger-codegen.git

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

class seqann.feature_client.models.feature_request.**FeatureRequest**(*locus=None*, *term=None*, *rank=None*, *sequence=None*)

Bases: object

NOTE: This class is auto generated by the swagger code generator program. Do not edit the class manually.

locus
Gets the locus of this FeatureRequest. locus name or URI

Returns The locus of this FeatureRequest.

Return type str

term
Gets the term of this FeatureRequest. Sequence Ontology (SO) term name, accession, or URI

Returns The term of this FeatureRequest.

Return type str

rank
Gets the rank of this FeatureRequest. feature rank, must be at least 1

Returns The rank of this FeatureRequest.

Return type int

sequence
Gets the sequence of this FeatureRequest. feature sequence, in DNA alphabet

Returns The sequence of this FeatureRequest.

Return type str

to_dict()
Returns the model properties as a dict

to_str()
Returns the string representation of the model

seqann.feature_client.models.sequence module

Module contents

No descripton provided (generated by Swagger Codegen https://github.com/swagger-api/swagger-codegen)

OpenAPI spec version: 1.0

Generated by: https://github.com/swagger-api/swagger-codegen.git

15.2 Submodules

15.3 seqann.feature_client.api_client module

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

ref: https://github.com/swagger-api/swagger-codegen

class seqann.feature_client.api_client.**ApiClient**(*host=None*, *header_name=None*, *header_value=None*, *cookie=None*)

Bases: object

Generic API client for Swagger client library builds.

Swagger generic API client. This client handles the client- server communication, and is invariant across implementations. Specifics of the methods and models for each application are generated from the Swagger templates.

NOTE: This class is auto generated by the swagger code generator program. Ref: https://github.com/swagger-api/swagger-codegen Do not edit the class manually.

Parameters

- **host** – The base path for the server to call.
- **header_name** – a header to pass when making calls to the API.
- **header_value** – a header value to pass when making calls to the API.

user_agent
Gets user agent.

set_default_header(*header_name*, *header_value*)

to_path_value(*obj*)
Takes value and turn it into a string suitable for inclusion in the path, by url-encoding.

Parameters **obj** – object or string value.

Return string quoted value.

sanitize_for_serialization(*obj*)
Builds a JSON POST object.

If obj is None, return None. If obj is str, int, long, float, bool, return directly. If obj is datetime.datetime, datetime.date

convert to string in iso8601 format.

If obj is list, sanitize each element in the list. If obj is dict, return the dict. If obj is swagger model, return the properties dict.

Parameters **obj** – The data to serialize.

Returns The serialized form of data.

deserialize(*response*, *response_type*)
Deserializes response into an object.

Parameters

- **response** – RESTResponse object to be deserialized.
- **response_type** – class literal for deserialzied object, or string of class name.

Returns deserialized object.

call_api(*resource_path*, *method*, *path_params=None*, *query_params=None*, *header_params=None*, *body=None*, *post_params=None*, *files=None*, *response_type=None*, *auth_settings=None*, *callback=None*, *_return_http_data_only=None*)
Makes the HTTP request (synchronous) and return the deserialized data. To make an async request, define a function for callback.

Parameters

- **resource_path** – Path to method endpoint.
- **method** – Method to call.
- **path_params** – Path parameters in the url.
- **query_params** – Query parameters in the url.
- **header_params** – Header parameters to be placed in the request header.
- **body** – Request body.
- **dict** (*files*) – Request post form parameters, for *application/x-www-form-urlencoded*, *multipart/form-data*.
- **list** (*auth_settings*) – Auth Settings names for the request.
- **response** – Response data type.
- **dict** – key -> filename, value -> filepath, for *multipart/form-data*.
- **function** (*callback*) – Callback function for asynchronous request. If provide this parameter, the request will be called asynchronously.
- **_return_http_data_only** – response data without head status code and headers

Returns If provide parameter callback, the request will be called asynchronously. The method will return the request thread. If parameter callback is None, then the method will return the response directly.

request(*method*, *url*, *query_params=None*, *headers=None*, *post_params=None*, *body=None*)
Makes the HTTP request using RESTClient.

prepare_post_parameters(*post_params=None*, *files=None*)
Builds form parameters.

Parameters

- **post_params** – Normal form parameters.
- **files** – File parameters.

Returns Form parameters with files.

select_header_accept(*accepts*)
Returns *Accept* based on an array of accepts provided.

Parameters **accepts** – List of headers.

Returns Accept (e.g. application/json).

select_header_content_type(*content_types*)
Returns *Content-Type* based on an array of content_types provided.

Parameters **content_types** – List of content-types.

Returns Content-Type (e.g. application/json).

update_params_for_auth(*headers*, *querys*, *auth_settings*)
Updates header and query params based on authentication setting.

Parameters

- **headers** – Header parameters dict to be updated.
- **querys** – Query parameters dict to be updated.
- **auth_settings** – Authentication setting identifiers list.

15.4 seqann.feature_client.configuration module

No descripton provided (generated by Swagger Codegen https://github.com/swagger-api/swagger-codegen)

OpenAPI spec version: 1.0

Generated by: https://github.com/swagger-api/swagger-codegen.git

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

seqann.feature_client.configuration.**singleton**(*cls*, **args*, ***kw*)

seqann.feature_client.configuration.**Configuration**()

15.5 seqann.feature_client.rest module

No descripton provided (generated by Swagger Codegen https://github.com/swagger-api/swagger-codegen)

OpenAPI spec version: 1.0

Generated by: https://github.com/swagger-api/swagger-codegen.git

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

class seqann.feature_client.rest.**RESTResponse**(*resp*)

Bases: io.IOBase

getheaders()

Returns a dictionary of the response headers.

getheader(*name*, *default=None*)

Returns a given response header.

class seqann.feature_client.rest.**RESTClientObject**(*pools_size=4*)

Bases: object

request(*method*, *url*, *query_params=None*, *headers=None*, *body=None*, *post_params=None*)

Parameters

- **method** – http request method
- **url** – http request url
- **query_params** – query parameters in the url
- **headers** – http request headers
- **body** – request json body, for *application/json*
- **post_params** – request post parameters, *application/x-www-form-urlencode* and *multipart/form-data*

GET(*url*, *headers=None*, *query_params=None*)

HEAD(*url*, *headers=None*, *query_params=None*)

OPTIONS(*url*, *headers=None*, *query_params=None*, *post_params=None*, *body=None*)

DELETE(*url*, *headers=None*, *query_params=None*, *body=None*)

POST(*url*, *headers=None*, *query_params=None*, *post_params=None*, *body=None*)

PUT(*url*, *headers=None*, *query_params=None*, *post_params=None*, *body=None*)

PATCH(*url*, *headers=None*, *query_params=None*, *post_params=None*, *body=None*)

exception seqann.feature_client.rest.**ApiException**(*status=None*, *reason=None*, *http_resp=None*)

Bases: Exception

15.6 Module contents

No descripton provided (generated by Swagger Codegen https://github.com/swagger-api/swagger-codegen)

OpenAPI spec version: 1.0

Generated by: https://github.com/swagger-api/swagger-codegen.git

CHAPTER 16

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

16.1 Types of Contributions

16.1.1 Report Bugs

Report bugs at https://github.com/nmdp-bioinformatics/SeqAnn/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

16.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

16.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

16.1.4 Write Documentation

SeqAnn could always use more documentation, whether as part of the official SeqAnn docs, in docstrings, or even on the web in blog posts, articles, and such.

16.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/mhalagan-nmdp/gfe/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

16.2 Get Started!

Ready to contribute? Here's how to set up *gfe* for local development.

1. Fork the *gfe* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/SeqAnn.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv SeqAnn
$ cd SeqAnn/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 gfe tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

16.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, 3.3, 3.4 and 3.5, and for PyPy. Check https://travis-ci.org/nmdp-bioinformatics/SeqAnn/pull_requests and make sure that the tests pass for all supported Python versions.

16.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_seqann.TestBioSeqAnn.test_004_insertion
```

CHAPTER 17

History

17.1 0.0.30 (2018-06-14)

- Fixed issues with annotating class II sequences with large insertions (HLA-DRB1*04:04:01)

17.2 0.0.29 (2018-06-14)

- Fixed issue caused by annotating partial sequences with no locus provided

17.3 0.0.28 (2018-06-14)

- Running seqann without a BioSQL server is now substantially faster.

17.4 0.0.27 (2018-05-31)

- Made substantial improvements to

17.5 0.0.26 (2018-05-31)

-

17.6 0.0.25 (2018-05-31)

-

17.7 0.0.24 (2018-05-31)

- Fixed issue with DPA1 not being in max structure dictionary
- Removed BioPython warnings

17.8 0.0.23 (2018-05-25)

- Added GFE creation
- Added **exact** variable to `Annotation` class

17.9 0.0.22 (2018-05-25)

- Fixed issue with logging numerical values

17.10 0.0.21 (2018-05-25)

- Fixed how the ref_align step was being done.
- The alignments are now done using the previous partial alignment instead of redoing the sequence search step.

17.11 0.0.1 (2017-10-19)

- First release on PyPI.

CHAPTER 18

Credits

18.1 Development Lead

- Mike Halagan <mhalagan@nmdp.org>

18.2 Contributors

- Mike Halagan <mhalagan@nmdp.org>
- Wei Wang <wwang@nmdp.org>
- Hu Huang <hhuang@nmdp.org>

CHAPTER 19

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index

G

H

K

L

M

O

P

R

S

T

U

V