

---

# **SeqAnn Documentation**

*Release 1.0.0*

**Mike Halagan**

**Sep 26, 2019**



<b>1</b>	<b>Overview</b>	<b>3</b>
<b>2</b>	<b>Install</b>	<b>5</b>
<b>3</b>	<b>Parameters</b>	<b>7</b>
<b>4</b>	<b>Usage</b>	<b>9</b>
<b>5</b>	<b>Annotations</b>	<b>11</b>
<b>6</b>	<b>Dependencies</b>	<b>13</b>
<b>7</b>	<b>Installation</b>	<b>15</b>
7.1	Stable release . . . . .	15
7.2	From sources . . . . .	15
<b>8</b>	<b>Annotation Algorithm</b>	<b>17</b>
<b>9</b>	<b>BioSQL Database</b>	<b>19</b>
9.1	Running Container . . . . .	19
9.2	Building Locally . . . . .	19
<b>10</b>	<b>Creating blastn files</b>	<b>21</b>
<b>11</b>	<b>Debugging</b>	<b>23</b>
<b>12</b>	<b>Testing</b>	<b>25</b>
12.1	Different tests . . . . .	25
12.2	Running specific tests . . . . .	25
<b>13</b>	<b>seqann package</b>	<b>27</b>
13.1	seqann.sequence_annotation . . . . .	27
13.2	seqann.seq_search . . . . .	30
13.3	seqann.gfe . . . . .	31
13.4	seqann.blast_cmd . . . . .	32
13.5	seqann.align . . . . .	32
<b>14</b>	<b>seqann models</b>	<b>35</b>

14.1	Annotation	35
14.2	Reference Data	37
14.3	Blast	40
<b>15</b>	<b>seqann.feature_client package</b>	<b>41</b>
15.1	Subpackages	41
15.2	Submodules	47
15.3	seqann.feature_client.api_client module	47
15.4	seqann.feature_client.configuration module	50
15.5	seqann.feature_client.rest module	50
15.6	Module contents	51
<b>16</b>	<b>Contributing</b>	<b>53</b>
16.1	Types of Contributions	53
16.2	Get Started!	54
16.3	Pull Request Guidelines	55
16.4	Tips	55
<b>17</b>	<b>History</b>	<b>57</b>
17.1	0.0.30 (2018-06-14)	57
17.2	0.0.29 (2018-06-14)	57
17.3	0.0.28 (2018-06-14)	57
17.4	0.0.27 (2018-05-31)	57
17.5	0.0.26 (2018-05-31)	57
17.6	0.0.25 (2018-05-31)	57
17.7	0.0.24 (2018-05-31)	58
17.8	0.0.23 (2018-05-25)	58
17.9	0.0.22 (2018-05-25)	58
17.10	0.0.21 (2018-05-25)	58
17.11	0.0.1 (2017-10-19)	58
<b>18</b>	<b>Credits</b>	<b>59</b>
18.1	Development Lead	59
18.2	Contributors	59
<b>19</b>	<b>Indices and tables</b>	<b>61</b>
	<b>Python Module Index</b>	<b>63</b>
	<b>Index</b>	<b>65</b>

Copyright (c) 2018 Be The Match operated by National Marrow Donor Program. All Rights Reserved.

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 initializing a `BioSeqAnn` class. However, annotations can be created significantly faster when using a BioSQL database. When a BioSQL database is not provided the latest `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 initializing a `BioSeqAnn` object.

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



---

## 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.



```
{
  '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(
↪ 'AGGATTTTCGTGTACCAGTTTAAGGCCATGTGCTACTTCACCAACGGGACGGAGC...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-Aw2-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-4',
  'seq': SeqRecord(seq=Seq('AGAGACTCTCCCGAGGATTTTCGTGTACCAGTTTAAGGCCATGTGCTACTTCACC.
↪ ..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



## 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
```



---

## 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

## 9.1 Running Container

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

```
$ docker pull nmdpbiinformatics/imgt_biosqldb
```

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

```
$ docker run -d --name imgt_biosqldb -p 3306:3306 \  
-e MYSQL_ROOT_PASSWORD=my-secret-pw nmdpbiinformatics/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 versions 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-biinformatics/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 processes. 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)
```



**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
```

## 13.1 seqann.sequence\_annotation

```
class seqann.sequence_annotation.BioSeqAnn(server: <module
    'BioSQL.BioSeqDatabase' from
    '/home/docs/checkouts/readthedocs.org/user_builds/seqann/envs/latest/
    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: se-
    qann.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 retrieving 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(
↪ 'AGGATTTTCGTGTACCAGTTTAAGGCCATGTGCTACTTCACCAACGGGACGGAGC...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-Aw2-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-4',
        'seq': SeqRecord(seq=Seq(
↳'AGAGACTCTCCCGAGGATTTTCGTGTACCAGTTTAAGGCCATGTGCTACTTCACC...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('AGAGACTCTCCCGAGGATTTTCGTGTACCAGTTTAAGGCCATGTGCTACTTCACC')
>>> 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
↳AGGATTTTCGTGTACCAGTTTAAGGCCATGTGCTACTTCACCAACGGGACGGAGCGCGTTATGTGACCAGATACATCTATAACCGAG
exon_3 nt_search and clustalo
↳TGGAGCCCACAGTGACCATCTCCCCATCCAGGACAGAGGCCCTCAACCACCACAACCTGCTGGTCTGCTCAGTGACAGATTCTATCCAG
exon_1 nt_search and clustalo AGAGACTCTCCCG
exon_4 nt_search and clustalo GGGCTCAGTCTGAATCTGCCAGAGCAAGATG

```

**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('AGAGACTCTCCCGAGGATTTTCGTGTACCAGTTTAAGGCCATGTGCTACTTCACC')
>>> 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
```

**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*, *value=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.
- **value** (*int*) – The evaluate 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('AGAGACTCTCCCGAGGATTTTCGTGTACCAGTTTAAGGCCATGTGCTACTTCACC')
>>> blast = blastn(sequence, locus, nseqs)
```

seqann.blast\_cmd.**get\_locus**(*sequences*, *kir=False*, *verbose=False*, *refdata=None*, *value=10*)

Gets the locus of the sequence by running blastn

### Parameters

- **sequences** – sequencences 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('AGAGACTCTCCCGAGGATTTTCGTGTACCAGTTTAAGGCCATGTGCTACTTCACC')
>>> 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*) – dictionary 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*) – Dictionary of sequence features
- **sequ** (*List*) – The sequence alignment for the input sequence
- **annotated** (*dict*) – dictionary 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`) – Dictionary 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`

## 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/environments/
    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('AGAGACTCTCCCGAGGATTTTCGTGTACCAGTTTAAGGCCATGTGCTACTTCACC')
>>> 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

---

## seqann.feature\_client package

---

### 15.1 Subpackages

#### 15.1.1 seqann.feature\_client.apis package

##### Submodules

##### seqann.feature\_client.apis.features\_api module

No description 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 description 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.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 description 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 description 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.

## 15.2 Submodules

### 15.3 seqann.feature\_client.api\_client module

Copyright 2016 SmartBear Software

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.

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-urlencoded* 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>

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.





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](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
```



### **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.

### 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`



**S**

`seqann.align`, 32  
`seqann.blast_cmd`, 32  
`seqann.feature_client`, 51  
`seqann.feature_client.api_client`, 47  
`seqann.feature_client.apis`, 45  
`seqann.feature_client.apis.features_api`,  
41  
`seqann.feature_client.configuration`, 50  
`seqann.feature_client.models`, 47  
`seqann.feature_client.models.feature`,  
45  
`seqann.feature_client.models.feature_request`,  
46  
`seqann.feature_client.rest`, 50  
`seqann.gfe`, 31  
`seqann.models.annotation`, 35  
`seqann.models.blast`, 40  
`seqann.models.reference_data`, 37  
`seqann.seq_search`, 30  
`seqann.sequence_annotation`, 27



## A

- accession (*seqann.feature\_client.models.feature.Feature attribute*), 46
- add\_alignment() (*seqann.sequence\_annotation.BioSeqAnn method*), 29
- align\_seqs() (*in module seqann.align*), 32
- aligned (*seqann.models.annotation.Annotation attribute*), 37
- alignments (*seqann.models.reference\_data.ReferenceData attribute*), 38
- alleles (*seqann.models.blast.Blast attribute*), 40
- ambig (*seqann.models.annotation.Annotation attribute*), 36
- annotate() (*seqann.sequence\_annotation.BioSeqAnn method*), 28
- Annotation (*class in seqann.models.annotation*), 35
- annotation (*seqann.models.annotation.Annotation attribute*), 37
- ApiClient (*class in seqann.feature\_client.api\_client*), 48
- ApiException, 51
- B**
- BioSeqAnn (*class in seqann.sequence\_annotation*), 27
- Blast (*class in seqann.models.blast*), 40
- blastdb (*seqann.models.reference\_data.ReferenceData attribute*), 39
- blastn() (*in module seqann.blast\_cmd*), 32
- blocks (*seqann.models.annotation.Annotation attribute*), 37
- C**
- call\_api() (*seqann.feature\_client.api\_client.ApiClient method*), 48
- check\_annotation() (*seqann.models.annotation.Annotation method*), 37
- clean() (*seqann.models.annotation.Annotation method*), 37
- complete\_annotation (*seqann.models.annotation.Annotation attribute*), 35
- Configuration() (*in module seqann.feature\_client.configuration*), 50
- count\_diffs() (*in module seqann.align*), 33
- covered (*seqann.models.annotation.Annotation attribute*), 36
- create\_feature() (*seqann.feature\_client.apis.features\_api.FeaturesApi method*), 41
- create\_feature\_with\_http\_info() (*seqann.feature\_client.apis.features\_api.FeaturesApi method*), 42
- D**
- datafile (*seqann.models.reference\_data.ReferenceData attribute*), 38
- dbversion (*seqann.models.reference\_data.ReferenceData attribute*), 38
- DELETE() (*seqann.feature\_client.rest.RESTClientObject method*), 51
- deserialize() (*seqann.feature\_client.api\_client.ApiClient method*), 48
- download\_dat() (*in module seqann.models.reference\_data*), 37
- E**
- exact (*seqann.models.annotation.Annotation attribute*), 35
- exact\_match (*seqann.models.annotation.Annotation attribute*), 37
- F**
- failed (*seqann.models.blast.Blast attribute*), 40
- Feature (*class in seqann.feature\_client.models.feature*), 45

- feature\_lengths (seqann.models.reference\_data.ReferenceData attribute), 39
- FeatureRequest (class in seqann.feature\_client.models.feature\_request), 46
- features (seqann.models.annotation.Annotation attribute), 36
- FeaturesApi (class in seqann.feature\_client.apis.features\_api), 41
- find\_features() (in module seqann.align), 33
- flatten() (in module seqann.align), 32
- from\_dict() (seqann.models.annotation.Annotation class method), 35
- from\_dict() (seqann.models.blast.Blast class method), 40
- from\_dict() (seqann.models.reference\_data.ReferenceData class method), 37
- from\_dict() (seqann.seq\_search.SeqSearch class method), 30
- ## G
- GET() (seqann.feature\_client.rest.RESTClientObject method), 51
- get\_feature\_by\_path() (seqann.feature\_client.apis.features\_api.FeaturesApi method), 42
- get\_feature\_by\_path\_with\_http\_info() (seqann.feature\_client.apis.features\_api.FeaturesApi method), 42
- get\_feature\_by\_query() (seqann.feature\_client.apis.features\_api.FeaturesApi method), 43
- get\_feature\_by\_query\_with\_http\_info() (seqann.feature\_client.apis.features\_api.FeaturesApi method), 43
- get\_gfe() (seqann.gfe.GFE method), 31
- get\_locus() (in module seqann.blast\_cmd), 32
- getblocks() (in module seqann.seq\_search), 31
- getblocks() (in module seqann.sequence\_annotation), 29
- getheader() (seqann.feature\_client.rest.RESTResponse method), 50
- getheaders() (seqann.feature\_client.rest.RESTResponse method), 50
- GFE (class in seqann.gfe), 31
- gfe (seqann.models.annotation.Annotation attribute), 37
- ## H
- has\_hla() (in module seqann.blast\_cmd), 32
- hash\_code (seqann.feature\_client.models.feature.Feature attribute), 46
- HEAD() (seqann.feature\_client.rest.RESTClientObject method), 51
- hla\_loci (seqann.models.reference\_data.ReferenceData attribute), 39
- hla\_names (seqann.models.reference\_data.ReferenceData attribute), 39
- hlaref (seqann.models.reference\_data.ReferenceData attribute), 38
- ## K
- kir (seqann.models.reference\_data.ReferenceData attribute), 39
- ## L
- list\_features() (seqann.feature\_client.apis.features\_api.FeaturesApi method), 43
- list\_features\_0() (seqann.feature\_client.apis.features\_api.FeaturesApi method), 44
- list\_features\_0\_with\_http\_info() (seqann.feature\_client.apis.features\_api.FeaturesApi method), 44
- list\_features\_1() (seqann.feature\_client.apis.features\_api.FeaturesApi method), 44
- list\_features\_1\_with\_http\_info() (seqann.feature\_client.apis.features\_api.FeaturesApi method), 45
- list\_features\_with\_http\_info() (seqann.feature\_client.apis.features\_api.FeaturesApi method), 43
- load\_features() (seqann.gfe.GFE method), 31
- loctype() (in module seqann.seq\_search), 31
- locus (seqann.feature\_client.models.feature.Feature attribute), 45
- locus (seqann.feature\_client.models.feature\_request.FeatureRequest attribute), 46
- locus\_features() (seqann.gfe.GFE method), 31
- ## M
- mapping (seqann.models.annotation.Annotation attribute), 36
- match\_seqs (seqann.models.blast.Blast attribute), 40
- method (seqann.models.annotation.Annotation attribute), 36
- missing (seqann.models.annotation.Annotation attribute), 36
- ## O
- OPTIONS() (seqann.feature\_client.rest.RESTClientObject method), 51
- ## P
- PATCH() (seqann.feature\_client.rest.RESTClientObject method), 51

- POST () (*seqann.feature\_client.rest.RESTClientObject method*), 51
- prepare\_post\_parameters () (*seqann.feature\_client.api\_client.ApiClient method*), 49
- PUT () (*seqann.feature\_client.rest.RESTClientObject method*), 51
- ## R
- rank (*seqann.feature\_client.models.feature.Feature attribute*), 46
- rank (*seqann.feature\_client.models.feature\_request.FeatureRequest attribute*), 47
- ref\_align () (*seqann.sequence\_annotation.BioSeqAnn method*), 29
- ReferenceData (class in *seqann.models.reference\_data*), 37
- refmissing (*seqann.models.annotation.Annotation attribute*), 36
- request () (*seqann.feature\_client.api\_client.ApiClient method*), 49
- request () (*seqann.feature\_client.rest.RESTClientObject method*), 50
- resolve\_feats () (in module *seqann.align*), 33
- RESTClientObject (class in *seqann.feature\_client.rest*), 50
- RESTResponse (class in *seqann.feature\_client.rest*), 50
- ## S
- sanitize\_for\_serialization () (*seqann.feature\_client.api\_client.ApiClient method*), 48
- search\_refdata () (*seqann.models.reference\_data.ReferenceData method*), 39
- search\_seqs () (*seqann.seq\_search.SeqSearch method*), 30
- select\_header\_accept () (*seqann.feature\_client.api\_client.ApiClient method*), 49
- select\_header\_content\_type () (*seqann.feature\_client.api\_client.ApiClient method*), 49
- seq (*seqann.models.annotation.Annotation attribute*), 36
- seqann.align (module), 32
- seqann.blast\_cmd (module), 32
- seqann.feature\_client (module), 51
- seqann.feature\_client.api\_client (module), 47
- seqann.feature\_client.apis (module), 45
- seqann.feature\_client.apis.features\_api (module), 41
- seqann.feature\_client.configuration (module), 50
- seqann.feature\_client.models (module), 47
- seqann.feature\_client.models.feature (module), 45
- seqann.feature\_client.models.feature\_request (module), 46
- seqann.feature\_client.rest (module), 50
- seqann.gfe (module), 31
- seqann.models.annotation (module), 35
- seqann.models.blast (module), 40
- seqann.models.reference\_data (module), 37
- seqann.seq\_search (module), 30
- seqann.sequence\_annotation (module), 27
- seqannotation () (*seqann.models.reference\_data.ReferenceData method*), 40
- seqrecord () (*seqann.models.reference\_data.ReferenceData method*), 40
- seqref (*seqann.models.reference\_data.ReferenceData attribute*), 39
- SeqSearch (class in *seqann.seq\_search*), 30
- sequence (*seqann.feature\_client.models.feature.Feature attribute*), 46
- sequence (*seqann.feature\_client.models.feature\_request.FeatureRequest attribute*), 47
- server (*seqann.models.reference\_data.ReferenceData attribute*), 38
- server\_avail (*seqann.models.reference\_data.ReferenceData attribute*), 39
- set\_default\_header () (*seqann.feature\_client.api\_client.ApiClient method*), 48
- singleton () (in module *seqann.feature\_client.configuration*), 50
- struct\_order (*seqann.models.reference\_data.ReferenceData attribute*), 39
- structure (*seqann.models.annotation.Annotation attribute*), 36
- structure\_max (*seqann.models.reference\_data.ReferenceData attribute*), 38
- structures (*seqann.models.reference\_data.ReferenceData attribute*), 38
- ## T
- term (*seqann.feature\_client.models.feature.Feature attribute*), 45
- term (*seqann.feature\_client.models.feature\_request.FeatureRequest attribute*), 47
- to\_dict () (*seqann.feature\_client.models.feature.Feature method*), 46
- to\_dict () (*seqann.feature\_client.models.feature\_request.FeatureRequest method*), 47

`to_path_value()` (*seqann.feature\_client.api\_client.ApiClient* method), 48  
`to_str()` (*seqann.feature\_client.models.feature.Feature* method), 46  
`to_str()` (*seqann.feature\_client.models.feature\_request.FeatureRequest* method), 47

## U

`update_params_for_auth()` (*seqann.feature\_client.api\_client.ApiClient* method), 49  
`user_agent` (*seqann.feature\_client.api\_client.ApiClient* attribute), 48

## V

`verbose` (*seqann.models.reference\_data.ReferenceData* attribute), 38  
`verbose` (*seqann.seq\_search.SeqSearch* attribute), 30  
`verbosity` (*seqann.models.reference\_data.ReferenceData* attribute), 38  
`verbosity` (*seqann.seq\_search.SeqSearch* attribute), 30