CathPy Documentation

Release 0.1.4

Ian Sillitoe

Apr 16, 2019

Contents

Warning: This code is still in early release and may change.

CHAPTER 1

Guide

1.1 API

1.1.1 cathpy.align

```
from cathpy.align import Align

aln = Align.new_from_stockholm('/path/to/align.sto')

aln.count_sequences
# 75

seq = aln.find_seq_by_accession('1cukA01')
seq = aln.find_seq_by_id('1cukA01/1-151')
```

cathpy.align - manipulating protein sequences and alignments

class cathpy.align.**Align**(*seqs=None*, ***, *_id=None*, *accession=None*, *description=None*, *aln_type=None*, *min_bitscore=None*, *cath_version=None*, *dops_score=None*)

Object representing a protein sequence alignment.

add_groupsim()

Add groupsim annotation to this alignment.

add_scorecons()

Add scorecons annotation to this alignment.

add_sequence(*seq: cathpy.align.Sequence*)

Add a sequence to this alignment.

aln_positions

Returns the number of alignment positions.

copy()

Return a deepcopy of this object.

count_sequences
Returns the number of sequences in the alignment.

find_first_seq_by_accession(*acc*)
Returns the first Sequence with the given accession.

find_seq_by_accession(*acc*)
Returns the Sequence corresponding to the provided id.

find_seq_by_id(*_id*)
Returns the Sequence corresponding to the provided id.

get_seq_at_offset(*offset*)
Returns the Sequence at the given offset (zero-based).

id
Returns the id of this Align object.

insert_gap_at_offset(*offset*, *gap_char='-'*)
Insert a gap char at the given offset (zero-based).

lower_case_at_offset(*start*, *end=None*)
Lower case all the residues in the given alignment window.

merge_alignment(*merge_aln*, *ref_seq_acc: str*, *ref_correspondence: cathpy.align.Correspondence = None*, ***, *cluster_label=None*, *merge_ref_id=False*, *self_ref_id=False*)
Merges aligned sequences into the current object via a reference sequence.

Sequences in `merge_aln` are brought into the current alignment using the equivalences identified in reference sequence `ref_seq_acc` (which must exist in both the `self` and `merge_aln`).

This function was originally written to merge FunFam alignments according to structural equivalences identified by CORA (a multiple structural alignment tool). Moving between structure and sequence provides the added complication that sequences in the structural alignment (CORA) are based on ATOM records, whereas sequences in the merge alignment (FunFams) are based on SEQRES records. The `ref_correspondence` argument allows this mapping to be taken into account.

Parameters

- **merge_aln** (`Align`) – An Align containing the reference sequence and any additional sequences to merge.
- **ref_seq_acc** (*str*) – The accession that will be used to find the reference sequence in the current alignment and merge_aln
- **ref_correspondence** (`Correspondence`) – An optional Correspondence object that provides a mapping between the reference sequence found in `self` (ATOM records) and reference sequence as it appears in `merge_aln` (SEQRES records).
- **cluster_label** (*str*) – Provide a label to differentiate the sequences being merged (eg for groupsim calculations). A default label is provided if this is `None`.
- **self_ref_id** (*str*) – Specify the id to use when adding the ref sequence from the current alignment.
- **merge_ref_id** (*str*) – Specify the id to use when adding the ref sequence from the merge alignment. By default this sequence is only inluded in the final alignment (as `<id>_merge`) if a custom correspondence is provided.

Returns Array of Sequences added to the current alignment.

Return type [*Sequence*]

Raises ***MergeCorrespondenceError*** – problem mapping reference sequence between alignment and correspondence

classmethod new_from_fasta(*fasta_io*)
Initialises an alignment object from a FASTA file / string / io

classmethod new_from_pir(*pir_io*)
Initialises an alignment object from a PIR file / string / io

classmethod new_from_stockholm(*sto_io*, *, *nowarnings=False*)
Initialises an alignment object from a STOCKHOLM file / string / io

read_sequences_from_fasta(*fasta_io*)
Parses aligned sequences from FASTA (str, file, io) and adds them to the current Align object. Returns the number of sequences that are added.

read_sequences_from_pir(*pir_io*)
Parse aligned sequences from PIR (str, file, io) and adds them to the current Align object. Returns the number of sequences that are added.

remove_alignment_gaps()
Return a new alignment after removing alignment positions that contain a gap for all sequences.

remove_sequence_by_id(*seq_id: str*)
Removes a sequence from the alignment.

sequences
Provides access to the Sequence objects in the alignment.

set_gap_char_at_offset(*offset*, *gap_char*)
Override the gap char for all sequences at a given offset.

set_id(*_id*)
Sets the id of this Align object.

slice_seqs(*start*, *end=None*)
Return an array of Sequence objects from start to end.

to_fasta(*wrap_width=80*)
Returns the alignment as a string (FASTA format)

to_pir(*wrap_width=80*)
Returns the alignment as a string (PIR format)

total_gap_positions
Returns the total number of gaps in the alignment.

total_positions
Returns the total number of positions in the alignment.

write_fasta(*fasta_file*, *wrap_width=80*)
Write the alignment to a file in FASTA format.

write_pir(*pir_file*, *wrap_width=80*, *, *use_accession=False*)
Write the alignment to a file in PIR format.

write_sto(*sto_file*, *, *meta=None*)
Write the alignment to a file in STOCKHOLM format.

class cathpy.align.**Correspondence**(*_id=None*, *residues=None*, ***kwargs*)
Provides a mapping between ATOM and SEQRES residues.

A correspondence is a type of alignment that provides the equivalences between the residues in the protein sequence (eg `SEQRES` records) and the residues actually observed in the structure (eg `ATOM` records).

Within CATH, this is most commonly initialised from a GCF file:

```
` aln = Correspondence.new_from_gcf('/path/to/<id>.gcf') `
```

TODO: allow this to be created from PDBe API endpoint.

apply_seqres_segments(*segs*)
Returns a new correspondence from just the residues within the segments.

atom_length
Return the number of ATOM residues

atom_sequence
Returns a Sequence corresponding to the ATOM records.

first_residue
Returns the first residue in the correspondence.

get_res_at_offset(*offset: int*)
Return the `Residue` at the given offset (zero-based)

get_res_by_atom_pos(*pos*)
Returns Residue corresponding to position in the ATOM sequence (ignores gaps).

get_res_by_pdb_label(*pdb_label*)
Returns the Residue that matches *pdb_label*

get_res_by_seq_num(*seq_num: int*)
Return the `Residue` with the given sequence number

get_res_offset_by_atom_pos(*pos*)
Returns offset of Residue at position in the ATOM sequence (ignores gaps).

last_residue
Returns the last residue in the correspondence.

classmethod new_from_gcf(*gcf_io*)
Create a new Correspondence object from a GCF io / filename / string.

This provides a correspondence between SEQRES and ATOM records for a given protein structure.

Example format:

> >gi|void|ref1 A 1 5 A K 2 6 K G 3 7 G H 4 8 H P 5 9 P G 6 10 G P 7 10A P K 8 10B K A 9 11 A P 10 * * G 11 * * ...

seqres_length
Return the number of SEQRES residues

seqres_sequence
Returns a Sequence corresponding to the SEQRES records.

set_id(*_id*)
Sets the id of the current Correspondence object

to_aln()
Returns the Correspondence as an Align object.

to_fasta(***kwargs*)
Returns the Correspondence as a string (FASTA format).

to_gcf()
Renders the current object as a GCF string.

Example format:

>gi|void|ref1 A 1 5 A K 2 6 K G 3 7 G H 4 8 H P 5 9 P G 6 10 G P 7 10A P K 8 10B K A 9 11 A P 10 * * G 11 * * ...

to_sequences()
Returns the Correspondence as a list of *Sequence* objects

class cathpy.align.**Sequence**(*hdr: str*, *seq: str*, ***, *meta=None*, *description=None*)
Class to represent a protein sequence.

accession_and_seginfo
Returns accession and segment info for this Sequence.

cluster_id
Returns the cluster id for this Sequence.

copy()
Provide a deep copy of this sequence.

get_offset_at_seq_position(*seq_pos*)
Return the offset (with gaps) of the given sequence position (ignores gaps).

get_res_at_offset(*offset*)
Return the residue character at the given offset (includes gaps).

get_res_at_seq_position(*seq_pos*)
Return the residue character at the given sequence position (ignores gaps).

get_residues()
Returns an array of Residue objects based on this sequence.

Note: if segment information has been specified then this will be used to calculate the *seq_num* attribute.

Raises ***OutOfBoundsError*** – problem mapping segment info to sequence

get_seq_position_at_offset(*offset*)
Returns sequence position (ignoring gaps) of the given residue (may include gaps).

id
Returns the id for this Sequence

insert_gap_at_offset(*offset*, *gap_char='-'*)
Insert a gap into the current sequence at a given offset.

is_cath_domain
Returns whether this Sequence is a CATH domain.

static is_gap(*res_char*)
Test whether a character is considered a gap.

length()
Return the length of the sequence.

lower_case_at_offset(*start*, *end=None*)
Lower case the residues in the given sequence window.

seq
Return the amino acid sequence as a string.

seq_no_gaps
Return the amino acid sequence as a string (after removing all gaps).

set_all_gap_chars(*gap_char='-'*)
Sets all gap characters.

set_cluster_id(*id_str*)
Sets the cluster id for this Sequence.

set_gap_char_at_offset(*offset*, *gap_char*)
Set the gap character at the given offset.

If the residue at a given position is a gap, then override the gap char with the given character.

set_id(*_id*)
Sets the id of the current Sequence object

set_lower_case_to_gap(*gap_char='-'*)
Set all lower-case characters to gap.

slice_seq(*start*, *end=None*)
Return a slice of this sequence.

classmethod split_hdr(*hdr: str*) → dict
Splits a sequence header into meta information.

Parameters **hdr** (*str*) – header string (eg *'domain|4_2_0|1cukA01/3-23_56-123'*)

Returns

header info

{ 'id': 'domain|4_2_0|1cukA01/3-23_56-123', 'accession': '1cukA01', 'id_type': 'domain', 'id_ver': '4_2_0', 'segs': [Segment(3, 23), Segment(56,123)], 'meta': {}

}

Return type info (dict)

to_fasta(*wrap_width=80*)
Return a string for this Sequence in FASTA format.

to_pir(*wrap_width=60*, *use_accession=False*)
Return a string for this Sequence in PIR format.

1.1.2 cathpy.datafiles

```
from cathpy import datafiles

release = datafiles.ReleaseDir('v4.2')

release.get_file('chaingcf', '1cukA01')
# /cath/data/v4_2_0/chaingcf/1cukA.gcf
```

Access data files

class cathpy.datafiles.**AtomFastaFileType**
Represents a FASTA file type (registered as 'atomfasta').

class cathpy.datafiles.**CombsFastaFileType**
Represents a FASTA file type (registered as 'combsfasta').

class cathpy.datafiles.**GcfFileType**
Represents a GCF file type (registered as 'chaingcf').

class cathpy.datafiles.**GenericFileType**
Represents a type of CATH Data file.

class cathpy.datafiles.**ReleaseDir**(*cath_version*, *, *base_dir='/cath/data'*)
Provides access to files relating to an official release of CATH.

Parameters

- **cath_version** – version of CATH (eg 'v4_2_0')
- **base_dir** – root directory for all data files (default: '/cath/data')

get_file(*file_type*, *entity_id*)
Returns the path for the given file type and identifier.

Parameters

- **file_type** – type of file (eg 'chaingcf')
- **entity_id** – identifier of the CATH entity (eg '1cukA')

1.1.3 cathpy.error

```
from cathpy import error as err

raise err.OutOfBoundsError('error message')
```

CATH Exception Classes

exception cathpy.error.**DuplicateSequenceError**
More than one sequence in an alignment has the same id

exception cathpy.error.**FileEmptyError**
File is empty.

exception cathpy.error.**FileNotFoundError**
File not found.

exception cathpy.error.**GapError**
Exception raised when trying to find residue information about a gap position.

exception cathpy.error.**GeneralError**
General Exception class within the cathpy package.

exception cathpy.error.**HttpError**
Problem getting/sending data over HTTP

exception cathpy.error.**InvalidInputError**
Exception raised when an error is encountered due to incorrect input.

exception cathpy.error.**JsonError**
Problem parsing JSON

exception cathpy.error.**MergeCheckError**
Exception raised when an error is encountered when checking the merge.

exception cathpy.error.**MergeCorrespondenceError**(*, *seq_id*, *aln_type*, *seq_type*, *ref_no_gaps*, *corr_no_gaps*)
Exception raised when failing to match correspondence sequences during alignment merge.

exception cathpy.error.**MissingExecutableError**
Missing an external executable.

exception cathpy.error.**MissingGroupsimError**
Failed to find groupsim executable

exception cathpy.error.**MissingScoreconsError**
Failed to find scorecons executable

exception cathpy.error.**MissingSegmentsError**
Exception raised when segment information is missing.

exception cathpy.error.**NoMatchesError**
No matches.

exception cathpy.error.**OutOfBoundsError**
Exception raised when code has moved outside expected boundaries.

exception cathpy.error.**ParamError**
Incorrect parameters.

exception cathpy.error.**ParseError**
Failed to parse information.

exception cathpy.error.**SeqIOError**
General Exception class within the SeqIO module

exception cathpy.error.**TooManyMatchesError**
Found more matches than expected.

1.1.4 cathpy.funfhmmer

```
from cathpy.funfhmmer import Client

api = Client()

response = api.search_fasta(fasta_file='/path/to/seq.fa')

response.as_csv()
```

CATH FunFHMMER - tool for remote sequence search against CATH FunFams

class cathpy.funfhmmer.**ApiClientBase**(*base_url*, *, *default_accept='application/json'*)
Base class implementing default local behaviour of an API client.

get(*url*, *, *accept=None*)
Performs a GET request

post(*url*, *, *accept=None*)
Performs a POST request

class cathpy.funfhmmer.**CheckResponse**(*, *data*, *message*, *success*, **kwargs*)
Class that represents the response from FunFHMMER STATUS request.

class cathpy.funfhmmer.**Client**(*, *base_url='http://www.cathdb.info'*, *sleep=2*, *retries=50*, *log=None*)
Client for the CATH FunFhmmer API (protein sequence search server).

The CATH FunFhmmer server allows users to locate matching CATH Functional Families (FunFams) in their protein sequence.

check(*task_id*)
Checks the status of an existing search.

results(*task_id*)
Retrieves the results of a search.

search_fasta(*fasta=None*, *fasta_file=None*)
Submits a sequence search and retrieves results.

submit(*fasta*)
Submits a protein sequence to be searched and returns a task_id.

class cathpy.funfhmmer.**ResponseBase**(***kwargs*)
Base class that represents the HTTP response.

class cathpy.funfhmmer.**ResultResponse**(***, *query_fasta*, *funfam_scan*, *cath_version*, ***kwargs*)
Class that represents the response from FunFHMMER RESULTS request.

as_csv()
Returns the result as CSV

as_json(***, *pp=False*)
Returns the response as JSON formatter string.

class cathpy.funfhmmer.**SubmitResponse**(***kwargs*)
Class that represents the response from FunFHMMER SUBMIT request.

1.1.5 cathpy.models

Provides access to classes that representing general entities such as amino acids, db identifiers, etc.

```
from cathpy.models import (
    AminoAcid, AminoAcids,
    CathID, FunfamID,
    ClusterFile, )

aa = AminoAcids.get_by_id('A')

aa.one                       # 'A'
aa.three                     # 'ala'
aa.word                      # 'alanine'

AminoAcids.is_valid_aa('Z') # False

cathid = CathID("1.10.8.10.1")

cathid.sfam_id               # '1.10.8.10'
cathid.depth                 # 5
cathid.cath_id_to_depth(3)   # '1.10.8'

funfam_file = ClusterFile("/path/to/1.10.8.10-ff-1234.reduced.sto")

funfam_file.path             # '/path/to/'
funfam_file.sfam_id          # '1.10.8.10'
funfam_file.cluster_num      # 1234
funfam_file.cluster_type     # 'ff'
funfam_file.desc             # 'reduced'
funfam_file.suffix           # '.sto'
```

Collection of classes used to model CATH data

class cathpy.models.**AminoAcid**(*one*, *three*, *word*)
Class representing an Amino Acid.

class cathpy.models.**AminoAcids**
Provides access to recognised Amino Acids.

classmethod get_by_id(*aa_letter*)
Return the AminoAcid object by the given single character aa code.

classmethod is_valid_aa(*aa_letter*)
Check if aa is a valid single character aa code.

class cathpy.models.**CathID**(*cath_id*)
Represents a CATH ID.

cath_id
Returns the CATH ID as a string.

cath_id_to_depth(*depth*)
Returns the CATH ID as a string.

depth
Returns the depth of the CATH ID.

sfam_id
Returns the superfamily id of the CATH ID.

class cathpy.models.**ClusterFile**(*path*, ***, *dir=None*, *sfam_id=None*, *cluster_type=None*, *cluster_num=None*, *join_char=None*, *desc=None*, *suffix=None*)
Object that represents a file relating to a CATH Cluster.

eg.

/path/to/1.10.8.10-ff-1234.sto

classmethod split_path(*path*)
Returns information about a cluster based on the path (filename).

to_string(*join_char=None*)
Represents the ClusterFile as a string (file path).

class cathpy.models.**ClusterID**(*sfam_id*, *cluster_type*, *cluster_num*)
Represents a Cluster Identifier (FunFam, SC, etc)

classmethod new_from_file(*file*)
Parse a new ClusterID from a filename.

class cathpy.models.**FunfamID**(*sfam_id*, *cluster_num*)
Object that represents a Funfam ID.

class cathpy.models.**Residue**(*aa*, *seq_num=None*, *pdb_label=None*, ***, *pdb_aa=None*)
Class to represent a protein residue.

class cathpy.models.**Scan**(***, *results*, ***kwargs*)
Object to store a sequence scan.

class cathpy.models.**ScanHit**(***, *match_name*, *match_cath_id*, *match_description*, *match_length*, *hsps*, *significance*, *data*, ***kwargs*)
Object to store a hit from a sequence scan.

class cathpy.models.**ScanHsp**(***, *evalue*, *hit_start*, *hit_end*, *hit_string=None*, *homology_string=None*, *length*, *query_start*, *query_end*, *query_string=None*, *rank*, *score*, ***kwargs*)
Object to store the High Scoring Pair (HSP) from a sequence scan.

class cathpy.models.**ScanResult**(***, *query_name*, *hits*, ***kwargs*)
Object to store a result from a sequence scan.

class cathpy.models.**Segment**(*start: int*, *stop: int*)
Class to represent a protein segment.

1.1.6 cathpy.util

```
from cathpy import util
```

General utility classes and functions

class cathpy.util.**AlignmentSummary**(*, *path*, *dops*, *aln_length*, *seq_count*, *gap_count*, *total_positions*)
Stores summary information about an alignment.

class cathpy.util.**AlignmentSummaryRunner**(*, *aln_dir=None*, *aln_file=None*, *suffix='.sto'*, *skipempty=False*)
Provides a summary report for sequence alignment files.

Parameters

- **aln_dir** – input alignment directory
- **aln_file** – input alignment file
- **suffix** – filter alignments by suffix
- **skipempty** – skip empty files

class cathpy.util.**FunfamFileFinder**(*base_dir*, *, *ff_tmpl='__SFAM__-ff-__FF_NUM__.sto'*)
Finds a Funfam alignment file within a directory.

funfam_id_from_file(*ff_file*)
Extracts a FunfamID from the file name (based on the ff_tmpl)

search_by_domain_id(*domain_id*)
Return the filename of the FunFam alignment containing the domain id.

class cathpy.util.**GroupsimResult**(*, *scores=None*)
Represents the result from running the groupsim algorithm.

count_positions
Returns the number of positions in the groupsim result.

classmethod new_from_file(*gs_file*)
Create a new groupsim result from an output file.

classmethod new_from_io(*gs_io*, *, *maxscore=1*)
Create a new groupsim result from an io source.

class cathpy.util.**GroupsimRunner**(*, *groupsim_dir='/home/docs/checkouts/readthedocs.org/user_builds/cathpy/envs/stal*
packages/cathpy-0.1.4-py3.7.egg/cathpy/tools/GroupSim', *python2path='python2'*, *column_gap=0.3*, *group_gap=0.5*)
Object that provides a wrapper around groupsim.

run_alignment(*alignment*, *, *column_gap=None*, *group_gap=None*, *mclachlan=False*)
Runs groupsim against a given alignment.

class cathpy.util.**ScoreconsResult**(*, *dops*, *scores*)
Represents the results from running scorecons.

to_string
Returns the scorecons results as a string (one char per position).

class cathpy.util.**ScoreconsRunner**(*, *scorecons_path='/home/docs/checkouts/readthedocs.org/user_builds/cathpy/envs/packages/cathpy-0.1.4-py3.7.egg/cathpy/tools/linux-x86_64/scorecons'*, *matrix_path='/home/docs/checkouts/readthedocs.org/user_builds/cathpy/envs/packages/cathpy-0.1.4-py3.7.egg/cathpy/tools/data/PET91mod.mat2'*)

Runs scorecons for a given alignment.

run_alignment(*alignment*)

Runs scorecons on a given alignment.

run_fasta(*fasta_file*)

Returns scorecons data (ScoreconsResult) for the provided FASTA file.

Returns scorecons result

Return type result (*ScoreconsResult*)

run_stockholm(*sto_file*)

Returns scorecons data for the provided STOCKHOLM file.

Returns scorecons result

Return type result (*ScoreconsResult*)

class cathpy.util.**StructuralClusterMerger**(*, *cath_version*, *sc_file*, *ff_dir*, *out_fasta=None*, *out_sto=None*, *ff_tmpl='__SFAM__-ff-__FF_NUM__.sto'*, *add_groupsim=True*, *add_scorecons=True*, *cath_release=None*)

Merges FunFams based on a structure-based alignment of representative sequences.

Parameters

- **cath_version** – version of CATH
- **sc_file** – structure-based alignment (**.fa*) of funfam reps
- **ff_dir** – path of the funfam alignments (**.sto*) to merge
- **out_fasta** – file to write merged alignment (FASTA)
- **out_sto** – file to write merged alignment (STOCKHOLM)
- **ff_tmpl** – template used to find the funfam alignment files
- **add_groupsim** – add groupsim data (default: True)
- **add_scorecons** – add scorecons data (default: True)
- **cath_release** – specify custom release data directory

1.1.7 cathpy.version

```
from cathpy.version import CathVersion

cv = CathVersion("v4.2") # or "v4_2_0", "current"

cv.dirname
# "4_2_0"

cv.pg_dbname
# "cathdb_v4_2_0"

cv.is_current
# False
```

cathpy.version - manipulating database / release versions

class cathpy.version.**CathVersion**(**args*, ***kwargs*)
Object that represents a CATH version.

dirname
Return the version represented as a directory name (eg 'v4_2_0').

is_current
Returns whether the version corresponds to 'current' (eg HEAD)

join(*join_char='.'*)
Returns the version string (with an optional join_char).

classmethod new_from_string(*version_str*)
Create a new CathVersion object from a string.

pg_dbname
Return the version represented as a postgresql database (eg 'cathdb_v4_2_0').

to_string()
Returns the CATH version in string form.

1.2 Need Help

Problems? Please contact i.sillitoe@ucl.ac.uk

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

C

Index

H

I

J

L

M

N

O

P

R

S

T

W