AMAS_JGLAHE

Generated by Doxygen 1.9.8

1 AMAS 1

1.1 This fork: AMAS_JGLAHE 1
1.2 Installation 1
1.3 Command line interface 2
  1.3.1 Examples 2
    1.3.1.1 Concatenating alignments 3
    1.3.1.2 Getting alignment statistics 3
    1.3.1.3 Converting among formats 4
    1.3.1.4 Splitting alignment by partitions (updated partition parsing AMAS_JGLAHE) 4
    1.3.1.5 Translating a DNA alignment into aligned protein sequences 7
    1.3.1.6 Creating replicate data sets 7
    1.3.1.7 Removing taxa/sequences from alignment 8
    1.3.1.8 Checking if input is aligned 8
    1.3.1.9 TODO Metapartitions 8
1.4 TODO AMAS as a Python module 8

2 Namespace Index 11

2.1 Namespace List 11

3 Hierarchical Index 13

3.1 Class Hierarchy 13

4 Class Index 15

4.1 Class List 15

5 File Index 17

5.1 File List 17

6 Namespace Documentation 19

6.1 amas Namespace Reference 19
  6.1.1 Variable Documentation 19
    6.1.1.1 __all__ 19
    6.1.1.2 __author__ 19
    6.1.1.3 __email__ 19
    6.1.1.4 __version__ 20
6.2 amas.AMAS Namespace Reference 20
  6.2.1 Detailed Description 20
  6.2.2 Function Documentation 20
    6.2.2.1 main() 20
    6.2.2.2 proportion() 21
    6.2.2.3 run() 22

7 Class Documentation 23

7.1 amas.AMAS.Alignment Class Reference 23

7.1.1 Detailed Description 24
7.1.2 Constructor & Destructor Documentation 25
    7.1.2.1 __init__() 25
7.1.3 Member Function Documentation 25
    7.1.3.1 __str__() 25
    7.1.3.2 all_same() 25
    7.1.3.3 append_count() 26
    7.1.3.4 check_data_type() 26
    7.1.3.5 get_alignment_length() 27
    7.1.3.6 get_aln_input() 28
    7.1.3.7 get_char_summary() 28
    7.1.3.8 get_column() 29
    7.1.3.9 get_counts() 29
    7.1.3.10 get_counts_from_parsed() 30
    7.1.3.11 get_counts_from_seq() 31
    7.1.3.12 get_matrix_cells() 31
    7.1.3.13 get_missing() 31
    7.1.3.14 get_missing_from_parsed() 32
    7.1.3.15 get_missing_from_seq() 32
    7.1.3.16 get_missing_percent() 33
    7.1.3.17 get_missing_percent_from_seq() 33
    7.1.3.18 get_name() 34
    7.1.3.19 get_parsed_aln() 34
    7.1.3.20 get_parsimony_informative() 35
    7.1.3.21 get_prop_parsimony() 35
    7.1.3.22 get_prop_variable() 35
    7.1.3.23 get_site_no_missing_ambiguous() 36
    7.1.3.24 get_sites_no_missing_ambiguous() 36
    7.1.3.25 get_taxa_no() 37
    7.1.3.26 get_taxon_char_summary() 37
    7.1.3.27 get_trim_selection() 38
    7.1.3.28 get_variable() 39
    7.1.3.29 matrix_creator() 39
    7.1.3.30 replace_missing() 40
    7.1.3.31 summarize_alignment() 40
    7.1.3.32 summarize_alignment_by_taxa() 41
7.1.4 Member Data Documentation 42
    7.1.4.1 all_matrix_cells 42
    7.1.4.2 check 42
    7.1.4.3 data_type 42
    7.1.4.4 in_file 43
    7.1.4.5 in_format 43

7.1.4.6 length 43
7.1.4.7 matrix 43
7.1.4.8 missing 43
7.1.4.9 missing_records 43
7.1.4.10 no_missing_ambiguous 44
7.1.4.11 parsed_aln 44
7.1.4.12 parsimony_informative 44
7.1.4.13 prop_parsimony 44
7.1.4.14 prop_variable 44
7.1.4.15 variable_sites 44
7.2 amas.AMAS.AminoAcidAlignment Class Reference 45
7.2.1 Detailed Description 45
7.2.2 Member Function Documentation 46
7.2.2.1 get_summary() 46
7.2.2.2 get_taxa_summary() 46
7.2.3 Member Data Documentation 47
7.2.3.1 alphabet 47
7.2.3.2 missing_ambiguous_chars 47
7.2.3.3 missing_chars 47
7.2.3.4 non_alphabet 47
7.3 amas.AMAS.DNAAlignment Class Reference 48
7.3.1 Detailed Description 49
7.3.2 Member Function Documentation 49
7.3.2.1 get_atgc_content() 49
7.3.2.2 get_atgc_from_parsed() 50
7.3.2.3 get_atgc_from_seq() 50
7.3.2.4 get_list_from_atgc() 51
7.3.2.5 get_summary() 52
7.3.2.6 get_taxa_summary() 52
7.3.3 Member Data Documentation 53
7.3.3.1 alphabet 53
7.3.3.2 missing_ambiguous_chars 53
7.3.3.3 missing_chars 53
7.3.3.4 non_alphabet 53
7.4 amas.AMAS.FileHandler Class Reference 53
7.4.1 Detailed Description 54
7.4.2 Constructor & Destructor Documentation 54
7.4.2.1 __init__() 54
7.4.3 Member Function Documentation 54
7.4.3.1 __enter__() 54
7.4.3.2 __exit__() 54
7.4.3.3 get_file_name() 55

7.4.4 Member Data Documentation 55
  7.4.4.1 file_name 55
  7.4.4.2 in_file 55
7.5 amas.AMAS.FileParser Class Reference 55
7.5.1 Detailed Description 56
7.5.2 Constructor & Destructor Documentation 56
  7.5.2.1 __init__() 56
7.5.3 Member Function Documentation 56
  7.5.3.1 fasta_parse() 56
  7.5.3.2 nexus_interleaved_parse() 57
  7.5.3.3 nexus_parse() 58
  7.5.3.4 partitions_parse() 58
  7.5.3.5 phylip_interleaved_parse() 60
  7.5.3.6 phylip_parse() 60
  7.5.3.7 translate_ambiguous() 60
7.5.4 Member Data Documentation 61
  7.5.4.1 chars_match 61
  7.5.4.2 counter 61
  7.5.4.3 in_file 61
  7.5.4.4 in_file_lines 62
  7.5.4.5 matches 62
  7.5.4.6 name_match 62
  7.5.4.7 name_matches 62
  7.5.4.8 records 62
  7.5.4.9 seq_match `[1/2]` 62
  7.5.4.10 seq_match `[2/2]` 63
  7.5.4.11 seq_matches 63
  7.5.4.12 sequence 63
  7.5.4.13 sequences 63
  7.5.4.14 tax_chars_matches 63
  7.5.4.15 tax_match 63
  7.5.4.16 taxa 64
7.6 amas.AMAS.MetaAlignment Class Reference 64
7.6.1 Detailed Description 66
7.6.2 Constructor & Destructor Documentation 66
  7.6.2.1 __init__() 66
7.6.3 Member Function Documentation 69
  7.6.3.1 file_overwrite_error() 69
  7.6.3.2 get_alignment_name() 70
  7.6.3.3 get_alignment_name_no_ext() 71
  7.6.3.4 get_alignment_object() 71
  7.6.3.5 get_alignment_objects() 72

7.6.3.6 get_concatenated() 72
7.6.3.7 get_extension() 74
7.6.3.8 get_metapartition_extension() 74
7.6.3.9 get_parsed_alignments() 75
7.6.3.10 get_partitioned() 75
7.6.3.11 get_partitions() 76
7.6.3.12 get_replicate() 77
7.6.3.13 get_summaries() 77
7.6.3.14 get_taxon_summaries() 79
7.6.3.15 get_translated() 80
7.6.3.16 get_trimmed() 80
7.6.3.17 natural_sort() 81
7.6.3.18 print_fasta() 82
7.6.3.19 print_iqtree_nexus_partitions() 82
7.6.3.20 print_nexus() 83
7.6.3.21 print_nexus_int() 84
7.6.3.22 print_nexus_partitions() 85
7.6.3.23 print_phylip() 86
7.6.3.24 print_phylip_int() 87
7.6.3.25 print_raxml_partitions() 88
7.6.3.26 print_unspecified_partitions() 89
7.6.3.27 remove_empty_sequences() 90
7.6.3.28 remove_from_alignment() 91
7.6.3.29 remove_taxa() 91
7.6.3.30 remove_unknown_chars() 92
7.6.3.31 replace_string_in_file() 93
7.6.3.32 summarize_alignments() 93
7.6.3.33 summarize_alignments_taxa() 94
7.6.3.34 translate_dict() 94
7.6.3.35 translate_dna_to_aa() 95
7.6.3.36 trim_dict() 95
7.6.3.37 write_concat() 96
7.6.3.38 write_convert() 96
7.6.3.39 write_formatted_file() 97
7.6.3.40 write_metapartitions() 98
7.6.3.41 write_out() 99
7.6.3.42 write_partitions() 101
7.6.3.43 write_reduced() 102
7.6.3.44 write_replicate() 103
7.6.3.45 write_split() 104
7.6.3.46 write_summaries() 105
7.6.3.47 write_taxa_summaries() 105

7.6.3.48 write_translated() 106
7.6.3.49 write_trimmed() 106
7.6.4 Member Data Documentation 107
7.6.4.1 alignment_objects 107
7.6.4.2 by_taxon_summary 108
7.6.4.3 check_align 108
7.6.4.4 check_taxa 108
7.6.4.5 codes 108
7.6.4.6 codes_list 108
7.6.4.7 codons 108
7.6.4.8 command 108
7.6.4.9 concat_out 109
7.6.4.10 cores 109
7.6.4.11 data_type 109
7.6.4.12 gencode_NCBI_1 109
7.6.4.13 gencode_NCBI_10 109
7.6.4.14 gencode_NCBI_11 109
7.6.4.15 gencode_NCBI_12 110
7.6.4.16 gencode_NCBI_13 110
7.6.4.17 gencode_NCBI_14 110
7.6.4.18 gencode_NCBI_16 110
7.6.4.19 gencode_NCBI_2 110
7.6.4.20 gencode_NCBI_21 110
7.6.4.21 gencode_NCBI_22 110
7.6.4.22 gencode_NCBI_23 110
7.6.4.23 gencode_NCBI_24 111
7.6.4.24 gencode_NCBI_25 111
7.6.4.25 gencode_NCBI_26 111
7.6.4.26 gencode_NCBI_3 111
7.6.4.27 gencode_NCBI_4 111
7.6.4.28 gencode_NCBI_5 111
7.6.4.29 gencode_NCBI_6 111
7.6.4.30 gencode_NCBI_9 111
7.6.4.31 genetic_code 112
7.6.4.32 in_files 112
7.6.4.33 in_format 112
7.6.4.34 no_loci 112
7.6.4.35 no_mpan 112
7.6.4.36 no_replicates 112
7.6.4.37 no_sup_aln_name 113
7.6.4.38 parsed_alignments 113
7.6.4.39 parsimony_check 113

7.6.4.40 prepend_label 113
7.6.4.41 reading_frame 113
7.6.4.42 reduced_file_prefix 113
7.6.4.43 remove_empty 114
7.6.4.44 species_to_remove 114
7.6.4.45 species_to_remove_set 114
7.6.4.46 split 114
7.6.4.47 trim_fraction 114
7.6.4.48 trim_out 114
7.6.4.49 using_metapartitions 115
7.7 amas.AMAS.ParsedArgs Class Reference 115
7.7.1 Detailed Description 115
7.7.2 Constructor & Destructor Documentation 116
7.7.2.1 __init__() 116
7.7.3 Member Function Documentation 116
7.7.3.1 add_common_args() 116
7.7.3.2 concat() 117
7.7.3.3 convert() 118
7.7.3.4 get_args_dict() 119
7.7.3.5 metapartitions() 119
7.7.3.6 remove() 121
7.7.3.7 replicate() 121
7.7.3.8 split() 122
7.7.3.9 summary() 123
7.7.3.10 translate() 124
7.7.3.11 trim() 125
7.7.4 Member Data Documentation 126
7.7.4.1 args 126

8 File Documentation **127**
8.1 amas/__init__.py File Reference 127
8.2 __init__.py 127
8.3 amas/AMAS.py File Reference 127
8.4 AMAS.py 128
8.5 md/README.md File Reference 157

Index **159**

Chapter 1

AMAS

Alignment manipulation and summary statistics

If you are using this program, please cite this publication:

```
Borowiec, M.L. 2016. AMAS: a fast tool for alignment manipulation and computing of summary statistics. PeerJ
      4:e1660.
```

1.1 This fork: AMAS_JGLAHE

Fork of the main repo which:

- Adds a `metapartition` command -> collates discontinuous metapartitions within a superalignment and concatenates them into a new superalignment of contiguous metapartitons.
- Reduces restrictions on input partition file formatting -> accepts partition files for RAxML(-NG) and IQ-TREE2 (best_scheme, best_scheme.nex and best_model.nex).

1.2 Installation

Use `AMAS.py` in the `amas` directory as a stand-alone program or clone it if you have `git installed` on your system.

If your system doesn't have a Python version 3.4 or newer (`AMAS` will work under Python 3.0 but you may noy be able to use it with multiple cores), you will need to `download and install it`. On Linux-like systems (including Ubuntu) you can install it from the command line using

It may be possible to use this version as a module, but only through manual configuration.

1.3 Command line interface

`AMAS` can be run from the command line. Here is the general usage (you can view this in your command line with `python3 AMAS.py -h`):

```
usage: AMAS <command> [<args>]

The AMAS commands are:
  concat            Concatenate input alignments.
  convert           Convert to other file format.
  replicate         Create replicate data sets for phylogenetic jackknife.
  split             Split alignment according to a partitions file.
  summary           Write alignment summary.
  remove            Remove taxa from alignment.
  translate         Translate DNA alignment into protein alignment.
  trim              Remove columns from alignment.
  metapartitions    Runs `split` and concatenates the output.

Use AMAS <command> -h for help with arguments of the command of interest

positional arguments:
  command     Subcommand to run

optional arguments:
  -h, --help  show this help message and exit
```

To show help for individual commands, use `AMAS.py <command> -h` or `AMAS.py <command> --help`.

1.3.1 Examples

For every `AMAS.py` run on the command line you need to specify action with `concat`, `convert`, `replicate`, `split`, or `summary` for the input to be processed. Additionally, you need to provide three arguments required for all commands. The order in which the arguments are given does not matter:

1) input file name(s) with `-i` (or in long version: `--in-files`),

2) format with `-f` (`--in-format`),

3) and data type with `-d` (`--data-type`).

The options available for the format are `fasta`, `phylip`, `nexus` (sequential), `phylip-int`, and `nexus-int` (interleaved). Data types are `aa` for protein alignments and `dna` for nucleotide alignments.

For example:

```
python3 AMAS.py concat -i gene1.nex gene2.nex -f nexus -d dna
```

If you have many files that you want to input in one run, you can use multiple cores of your computer to process them in parallel. The `summary` command supports `-c` or `--cores` with which you can specify the number of cores to be used:

```
python3 AMAS.py summary -f phylip -d dna -i *phy -c 12
```

In the above, we specified 12 cores. Note that this won't improve computing time if you're working with only one or very few files. The parallel processing is only used for the file parsing step and calculating alignment summaries.

In addition to overall alignment summaries, you can also print statistics calculated on a sequence (taxon) by sequence basis. Use `-s` or `--by-taxon` flag to turn it on. `AMAS` in this mode will print out one file with overall alignment summaries and a file with taxon summaries for each input alignment.

IMPORTANT! `AMAS` is fast and powerful, but be careful: it assumes you know what you are doing and will not prevent you overwriting a file. It will, however, print out a warning if this has happened. `AMAS` was also written to work with aligned data and some of the output generated from unaligned sequences won't make sense. Because of computing efficiency `AMAS` by default does not check if input sequences are aligned. You can turn this option on with `-e` or `--check-align`.

1.3.1.1 Concatenating alignments

For example, if you want to concatenate all DNA phylip files in a directory and all of them have the `.phy` extension, you can run:

```
python3 AMAS.py concat -f phylip -d dna -i *phy
```

By default the output will be written to two files: `partitions.txt`, containing partitions from which your new alignment was constructed, and `concatenated.out` with the alignment itself in the fasta format. You can change the default names for these files with `-p` (`--concat-part`) and `-t` (`--concat-out`), respectively, followed by the desired name. The output format is specified by `-u` (`--out-format`) and can also be any of the following: `fasta`, `phylip`, `nexus` (sequential), `phylip-int`, or `nexus-int` (interleaved).

Below is a command specifying the concatenated file output format as nexus with `-u nexus`:

```
python3 AMAS.py concat -f fasta -d aa -i *fas -u nexus
```

Alignments to be concatenated need not have identical sets of taxa before processing: the concatenated alignment will be populated with missing data where a given locus is missing a taxon. However, if every file to be concatenated includes only unique names (for example species name plus sequence name: `D_melanogaster_NW↩_001845408.1` in one alignment, `D_melanogaster_NW_001848855.1` in other alignment etc.), you will first need to trim those names so that sequences from one taxon have equivalents in all files.
In addition to the name, you can also specify the format of the partitions output file. By default, the format is the following:

```
AA = 1-605
AK = 606-1200
28S = 1201-1800
```

RAxML:

```
python3 AMAS.py concat -f phylip -d dna -i *phy --part-format raxml
DNA, AA = 1-605
DNA, AK = 606-1200
DNA, 28S = 1201-1800
```

Nexus:

```
python3 AMAS.py concat -f phylip -d dna -i *phy --part-format nexus
#NEXUS
Begin sets;
    charset AA = 1-605;
    charset AK = 606-1200;
    charset 28S = 1201-1800;
End;
```

Partitions can also be written by codon positions using the `-n` or `--codons` flag, either for alignments containing first and second or all three positions. In the above example, supplying `-n 123` would result in:

```
AA_pos1 = 1-605\3
AA_pos2 = 2-605\3
AA_pos3 = 3-605\3
AK_pos1 = 606-1200\3
AK_pos2 = 607-1200\3
AK_pos3 = 608-1200\3
28S_pos1 = 1201-1800\3
28S_pos2 = 1202-1800\3
28S_pos3 = 1203-1800\3
```

1.3.1.2 Getting alignment statistics

This is an example of how you can summarize two protein fasta alignments by running:

```
python3 AMAS.py summary -f fasta -d aa -i my_aln.fasta my_aln2.fasta
```

By default `AMAS` will write a file with the summary of the alignment in `summary.txt`. You can change the name of this file with `-o` or `--summary-out`. You can also summarize a single or multiple sequence alignments at once.

The statistics calculated include the number of taxa, alignment length, total number of matrix cells, overall number of undetermined characters, percent of missing data, AT and GC contents (for DNA alignments), number and proportion of variable sites, number and proportion of parsimony informative sites, and counts of all characters present in the relevant alphabet.

1.3.1.3 Converting among formats

To convert all nucleotide fasta files with a `.fas` extension in a directory to nexus alignments, you could use:

```
python3 AMAS.py convert -d dna -f fasta -i *fas -u nexus
```

In the above, the required options are combined with `convert` command to convert the input files and `-u nexus` which indicates the output format.

`AMAS` will not overwrite over input here but will create new files instead, automatically appending appropriate extensions to the input file's name: `-out.fas`, `-out.phy`, `-out.int-phy`, `-out.nex`, or `-out.int-nex`.

1.3.1.4 Splitting alignment by partitions (updated partition parsing AMAS_JGLAHE)

If you have a partition file, you can split a concatenated alignment and write a file for each partition:

```
python3 AMAS.py split -f nexus -d dna -i concat.nex -l partitions.txt -u nexus
```

In the above one input file `concat.nex` was provided for splitting with `split` and partitions file `partitions.txt` with `-l` (same as `--split-by`). For splitting you should only use one input and one partition file at a time. This is an example partition file:

```
AApos1&2  =  1-604\3, 2-605\3
AApos3  =  3-606\3
28SAutapoInDels=7583, 7584, 7587, 7593
```

If this was the `partitions.txt` file from the example command above, `AMAS` would write three output files called `concat_AApos1&2.nex`, `concat_AApos3.nex`, and `concat_28SautapoInDels.nex`. The partitions file will be parsed correctly as long as there ~~is~~ are none ∼∼text prior to the partition name (`CHARSET AApos1&2` or `DNA, AApos1&2` will not work) and commas separate ranges or individual sites in each partition∼∼ of the following characters in partition names: `<>!?$^;:\/,.`; members of [\s] are also not allowed in partition names.

Sometimes after splitting you will have alignments with taxa that have only gaps `-` or missing data `?`. If you want to these to not be included in the output , add `-j` or `--remove-empty` to the command line.

Partition files are parsed with `AMAS.FileParser.partition_parse()`, which in the JGLAHE fork of AMAS looks like this:

```
def partitions_parse(self):
    # parse partitions file using regex
    # original: `matches = re.finditer(r"^(\s+)?([^ =]+)[ =]+([\\0-9, -]+)", self.in_file_lines,
      re.MULTILINE)`
    # new version: more permissive -> handles PartionFinder/RAxML/(IQ-TREE 2)best_scheme.nex format
      partition files
    matches = re.finditer(
        r"""^[ \t]*                              # start of line w/ zero-or-more (just) whitespaces/tabs
            (
             (?P<nexus>charset[ ]+)              # case 1: (IQ-TREE 2)best_scheme.nex partition
      directive; partition name
             |
             (?P<raxml>[A-Za-z0-9_+\.]+,[ \t]+)     # case 2: RAxML/RAxML-NG model(+other pars); partition
      name
            )?
            (?P<partition_name>[A-Za-z0-9_@*&()\[\]{}'"%#|-]+) # partition name (handles various
      metcharacters, excludes [\s],<>?/:;~`\^$!)
            [ ]*=[ ]*                            # whitespace-padded (or unpadded) '=': (IQ-TREE
      2)best_scheme.nex compatabiliy
            (?P<numbers>[\\0-9, -]+)             # position ranges w/ stride (multiple intervals; from
      original regex)
            (?P<nexus_term>[ ]*[;])?             # whitespace-prepended (or unprepended) ';' (nexus
      terminator)
        """,
        self.in_file_lines,
        re.MULTILINE | re.VERBOSE
    )

    # initiate list to store dictionaries with lists
    # of slice positions as values
    partitions = []
    add_to_partitions = partitions.append

    for match in matches:
        # initiate dictionary of partition name as key
        dict_of_dicts = {}
        # and list of dictionaries with slice positions
        list_of_dicts = []
        add_to_list_of_dicts = list_of_dicts.append
        # get parition name and numbers from parsed partition strings
        partition_name = match.group(’partition_name’)
        numbers = match.group(’numbers’)
        # remove any whitespace padding ’-’ (to be consistent with partition-writing format)
        numbers = re.sub(r"[ ]*-[ ]*", "-", numbers)
        # find all numbers that will be used to parse positions
        positions = re.findall(r"([^ ,]+)", numbers)

        for position in positions:
            # create dictionary for slicing input sequence
            # conditioning on whether positions are represented
            # by range, range with stride, or single number
            pos_dict = {}

            if "-" in position:
                m = re.search(r"([0-9]+)-([0-9]+)", position)
                pos_dict["start"] = int(m.group(1)) - 1
                pos_dict["stop"] = int(m.group(2))
            else:
                pos_dict["start"] = int(position) - 1
                pos_dict["stop"] = int(position)

            if "\\" in position:
                # Note: the value of `N‘ in `...\N‘ isn’t read: the script simply assumes `N‘ is consistent
  with the number of
                # increments per interval when the alignment is parsed with a stride of 3 (designating each
  cpos).
                # E.g. For the partition file:
                #       ...‘1-N\2‘
                #       ...‘2-N\2‘
                #       ...‘(N+1)-M\2‘
                #       ...‘(N+2)-M\2‘
                # 3’cpos are ignored due to the absence of intervals `3-N...‘, `(N+3)-M...‘, not because the
  associated stride values are‘\2‘
                pos_dict["stride"] = 3
            elif "\\" not in position:
                pos_dict["stride"] = 1

            add_to_list_of_dicts(pos_dict)

        dict_of_dicts[partition_name] = list_of_dicts
        add_to_partitions(dict_of_dicts)

    return partitions
```

This version removes some of the partition formatting restrictions of the original AMAS repo version, with the updated partitions_parse() method facilitating the uses of native RAxML(-NG) and IQ-TREE2 partition files.

The following is a contrived example to demonstrate this, along with the updated `metapartions` command. Consider the following superalignment `concat.fas`:

```
>S01
CCCCCTGGCGCCGCCGCCGCCCCTCCTCCTCCTCCCCCCCCC
>S02
AACAATGGAGCAGAAGAAGAAAATAATAATAATAATAAAAAAAAA
>S03
TTCTTTGGTGCTGATGTTGTTTTTTTTTTTTTTTTATTTTTTTTTT
>S04
GGCGGTGGGGCGGAGGTGGGGGGTGGTGGTGTTGATGGGGGGGGG
>S05
CCCCCTGGCGCCGACGTCGGCCCTCCTCGTCTTCATCCCCCCCCC
>S06
AACAATGGCGCAGAAGTAGGAAATACTAGTATTAATAAAAAAAAA
>S07
TTCTTTGGCGCAGATGTTGGTTATTCTTGTTTTTATTTTTTTTTT
>S08
GGCGTTGGCGCAGATGTGGGGGATGCTGGTGTTGATGGGGGGGGG
>S09
CGCCTTGGCGCAGATGTGGGCCATCCTCGTCTTCATCCCCCCCCC
>S10
AGCATTGGCGCAGATGTGGGCAATACTAGTATTCATAAAAAAAAA
>S11
TGCTTTGGCGCAGATGTGGGCTATTCTTGTATTCATTTTATTTTT
>S12
GGCGTTGGCGCAGATGTGGGCGATGCTTGTATTCATGGGAGGTGG
>S13
CGCCTTGGCGCAGATGTGGGCCATGCTTGTATTCATCCCAGCTCC
>S14
AGCATTGGCGCAGATGTGGGCCATGCTTGTATTCATAAAAGATCA
>S15
TGCATTGGCGCAGATGTGGGCCATGCTTGTATTCATTTTAGATCT
```

```
>S16
TGCATGTTCTCATATTTGTGCCAGGCGTGGATGCAGGG?AGATCT
>S17
TGCATGTTCTCATATTTGTGCCAGGCGTGGATGCAGTTTAGATCT
>S18
AGCATGTTCTCATATTTGTGCCAGGCGTGGATGCAGAAAAGATCA
>S19
CGCCTGTTCTCATATTTGTGCCAGGCGTGGATGCAGCCCAGCTCC
>S20
GGCGTGTTCTCATATTTGTGCGAGGCGTGGATGCAGGGGAGGTGG
>S21
TGCTTGTTCTCATATTTGTGCTAGTCGTGGATGCAGTTTATTTTT
>S22
AGCATGTTCTCATATTTGTGCAAGACGAGGATGCAGAAAAAAAAA
>S23
CGCCTGTTCTCATATTTGTGCCAGCCGCGGCTGCAGCCCCCCCCC
>S24
GGCGTGTTCTCATATTTGTGGGAGGCGGGGGTGGAGGGGGGGGGG
>S25
TTCTTGTTCTCATATTTTTGTTAGTCGTGGTTGTAGTTTTTTTTT
>S26
AACAAGTTCTCATAATTATGAAAGACGAGGATGAAGAAAAAAAAA
>S27
CCCCCGTTCTCCTACTTCTGCCCGCCGCGGCTGCAGCCCCCCCCC
>S28
GGCGGGTTGTCGTAGTTGTGGGGGGGGGGGGTGGAGGGGGGGGGG
>S29
TTCTTGTTTTCTTATTTTTTTTTGTTGTTGTTGTAGTTTTTTTTT
>S30
AACAAGTTATCATAATAATAAAAGAAGAAGAAGAAGAAAAAAAAA
>S31
CCCCCGTTCTCCTCCTCCTCCCCGCCGCCGCCGCCGCCCCCCCCC
```

Its partition file `partitions.txt` contain various formatting challenges to test the parser, but it includes the basic directives found in native RAxML(-NG) and IQ-TREE2 partition files:

```
charset @@(partition_A_pos1) =      7 - 21\3
  charset %s = 8  -21\3
    charset "partition_A_pos3" = 9-   21\3
Q.insect, (partition_B_pos)1 = 40-    45\3
Q.insect,   ’[partition_B_pos2] = 41   -45\3
Q.insect, par&&tition_B_pos3   =„  42-45\3
    GTR+G4,    pa&rtition_C =  , , , „38 39            37,
9.20b, #partition&|_D_pos1 = 1-6\3 22 - 36\3
       partition_D_pos2= 2-6\3 23 -36\3
p1_3_a                 = 3-6\3             24 -      36\3
  charpartition mymodels =
    9.20b: partition_D_pos1;
end;
```

While most of the partitions here are not contiguous with respect to the `concat.fas`, this superalignment can be converted into one with contiguous metapartitons by running the command:

```
./AMAS.py metapartitions -i concat.fas -f fasta -d dna --no-san --no-mpan -l partitions.txt -t
      concat.out.fas‘
```

This generates `concat.out.fas`:

```
>S01
GGGGGGCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCTTTTTT
>S02
GGGGGGCAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACTTTTTT
>S03
GGGGGGCATTTTTTTTTTTTTTTTTTTTTTTTTTTTTACTTTTTT
>S04
GGGGGGCATGGGGGGGGGGGGGGGGGGGGGGGGGGGTACTTTTTT
>S05
GGGGGGCATGCCCCCCCCCCCCCCCCCCCCCCCCCGTACTTTTTT
>S06
GGGGGGCATGCAAAAAAAAAAAAAAAAAAAAAAACGTACTTTTTT
>S07
GGGGGGCATGCATTTTTTTTTTTTTTTTTTTTTACGTACTTTTTT
>S08
GGGGGGCATGCATGGGGGGGGGGGGGGGGGGGTACGTACTTTTTT
>S09
GGGGGGCATGCATGCCCCCCCCCCCCCCCCCGTACGTACTTTTTT
>S10
GGGGGGCATGCATGCAAAAAAAAAAAAAAACGTACGTACTTTTTT
>S11
GGGGGGCATGCATGCATTTTTTTTTTTTTACGTACGTACTTTTTT
>S12
GGGGGGCATGCATGCATGGGGGGGGGGGTACGTACGTACTTTTTT
>S13
GGGGGGCATGCATGCATGCCCCCCCCCGTACGTACGTACTTTTTT
>S14
GGGGGGCATGCATGCATGCAAAAAAACGTACGTACGTACTTTTTT
```

```
>S15
GGGGGGCATGCATGCATGCATTTTTACGTACGTACGTACTTTTTT
>S16
TTTTTTCATGCATGCATGCATG?GTACGTACGTACGTACGGGGGG
>S17
TTTTTTCATGCATGCATGCATTTTTACGTACGTACGTACGGGGGG
>S18
TTTTTTCATGCATGCATGCAAAAAAACGTACGTACGTACGGGGGG
>S19
TTTTTTCATGCATGCATGCCCCCCCCCGTACGTACGTACGGGGGG
>S20
TTTTTTCATGCATGCATGGGGGGGGGGGTACGTACGTACGGGGGG
>S21
TTTTTTCATGCATGCATTTTTTTTTTTTTACGTACGTACGGGGGG
>S22
TTTTTTCATGCATGCAAAAAAAAAAAAAAACGTACGTACGGGGGG
>S23
TTTTTTCATGCATGCCCCCCCCCCCCCCCCCGTACGTACGGGGGG
>S24
TTTTTTCATGCATGGGGGGGGGGGGGGGGGGGTACGTACGGGGGG
>S25
TTTTTTCATGCATTTTTTTTTTTTTTTTTTTTTACGTACGGGGGG
>S26
TTTTTTCATGCAAAAAAAAAAAAAAAAAAAAAAACGTACGGGGGG
>S27
TTTTTTCATGCCCCCCCCCCCCCCCCCCCCCCCCCGTACGGGGGG
>S28
TTTTTTCATGGGGGGGGGGGGGGGGGGGGGGGGGGGTACGGGGGG
>S29
TTTTTTCATTTTTTTTTTTTTTTTTTTTTTTTTTTTTACGGGGGG
>S30
TTTTTTCAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACGGGGGG
>S31
TTTTTTCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCGGGGGG
```

The figure below more clearly demonstrates this transformation: `concat.fas` on the left is converted to `concat.out.fas` on the right using the `metapartitions` command above; visualized in `Aliview` v1.28

1.3.1.5 Translating a DNA alignment into aligned protein sequences

You can translate a nucleotide alignment to amino acids with AMAS using one of the `NCBI translation tables`. For example, to correctly translate an insect mitochondrial gene alignment that begins at a second codon position:

```
python3 AMAS.py translate -f nexus -d dna -i concat.nex --code 5 --reading-frame 2 --out-format phylip
```

`--code` and `--reading-frame` are the same as `-b` and `-k` and are both set to 1 (the standard genetic code and the first character of the alignment corresponds to the first codon position) by default. When translating, `AMAS` will contract gaps `-` and missing `?`, such that `---` becomes `-` in the translated alignment. A warning will be printed if stop codons are found and these are indicated as asterisks `*` in the output. See `AMAS.py translate -h` for more info.

1.3.1.6 Creating replicate data sets

With `AMAS` you can create concatenated alignments from a proportion of randomly chosen alignments that can be used for, for example, a phylogenetic jackknife analysis. Say you have 1000 phylip files, each containing a single aligned locus, and you want to create 200 replicate phylip alignments, each built from 100 loci randomly chosen from all the input files. You can do this by specifying `replicate` command and following it with `-r` or `--rep-aln` followed by the number of replicates (in this case `200`) and number of alignments (`100`). Remember to supply the output format with `-u` if you want it to be other than `fasta`:

```
python3 AMAS.py replicate -r 200 100 -d dna -f phylip -i *phy -u phylip
```

1.3.1.7 Removing taxa/sequences from alignment

It is possible to remove taxa from alignments:

```
python3 AMAS.py remove -x species1 species2 -d dna -f nexus -i *nex -u nexus-int -g no_species12_
```

The above will process all `nexus` files in the directory and remove taxa called `species1` and `species2`. The argument `-x` (the same as `--taxa-to-remove`) is followed by the names of sequences to be removed. Note that `AMAS` converts spaces into underscores and strips any quotes present in input sequence names before processing, so you may need to modify your names to remove accordingly. The argument `-g` (the same as `--out-prefix`) specifies a prefix to be added to output file names. The default prefix is 'reduced_'. You may want to realign your files after taxon removal.

1.3.1.8 Checking if input is aligned

By specifying optional argument `-e` (`--check-align`), you can make `AMAS` check if your input files contain only aligned sequences. This option is disabled by default because it can substantially increase computation times in files with many taxa. Enabling this option also provides an additional check against misspecified input file format.

1.3.1.9 TODO Metapartitions

1.4 TODO AMAS as a Python module

Using `AMAS` inside your Python pipeline gives you much more flexibility in how the input and output are being processed. All the major functions of the command line interface can recreated using `AMAS` as a module. Following installation from `pip` use:

```
pydoc amas.AMAS
```

To access detailed documentation for the classes and functions available.

You can import `AMAS` to your script with:

```
from amas import AMAS
```

The class used to manipulate alignments in `AMAS` is `MetaAlignment`. This class has to be instantiated with the same, named arguments as on the command line: `in_files`, `data_type`, `in_format`. You also need to supply the number of cores to be used with `cores`. MetaAlignment holds one or multiple alignments and its `in_files` option must be a list, even if only one file is being read.

```
meta_aln = AMAS.MetaAlignment(in_files=["gene1.phy"], data_type="dna",in_format="phylip", cores=1)
```

Creating MetaAlignment with multiple files is easy:

```
multi_meta_aln = AMAS.MetaAlignment(in_files=["gene1.phy", "gene1.phy"], data_type="dna",
      in_format="phylip", cores=2)
```

Now you can call the various methods on your alignments. `.get_summaries()` method will compute summaries for your alignments and produce headers for them as atuple with first element being the header and the second element a list of lists with the statistics:

```
summaries = meta_aln.get_summaries()
```

The header is different for nucleotide and amino acid data. You may choose to skip it and print only the second element of the tuple, that is a list of summary statistics:

```
= summaries[1]
```

`.get_parsed_alignments()` returns a list of dictionaries where each dictionary is an alignment and where taxa are the keys and sequences are the values. This allows you to, for example, print only taxa names in each alignment or do other manipulation of the sequence data:

```
# get parsed dictionaties
```

```
aln_dicts = multi_meta_aln.get_parsed_alignments()

# print only taxa names in the alignments:
for alignment in aln_dicts:
    for taxon_name in alignment.keys():
        print(taxon_name)
```

Similar to the above example, it is also easy to get translated amino acid alignment as a list of dictionaries (one per input alignment):

```
# get parsed dictionaties
aln_dicts = multi_meta_aln.get_translated(2, 1) # 2: vertebrate mitochondrial genetic code and 1: reading
      frame starting at first character
```

To split alignment use `.get_partitioned("your_partitions_file")` on a `MetaAlignment` with a single input file. `.get_partitioned()` returns a list of dictionaries of dictionaries, with `{ partition_name : { taxon : sequence } }` structure for each partition:

```
partitions = meta_aln.get_partitioned("partitions.txt")
```

`AMAS` uses `.get_partitions("your_partitions_file")` to parse the partition file:

```
parsed_parts = meta_aln.get_partitions("partitions.txt")
print(parsed_parts)
```

`.get_replicate(no_replicates, no_loci)` gives a list of parsed alignments (dictionaries), each a replicate constructed from the specified number of loci:

```
replicate_sets = multi_meta_aln.get_replicate(2, 2)
```

To concatenate multiple alignments first parse them with `.get_parsed_alignments()`, then pass to `.get_concatenated(your_parsed_alignments)`. This will return a tuple where the first element is the `{ taxon : sequence }` dictionary of concatenated alignment and the second element is the partitions dict with `{ name : range }`.

```
parsed_alns = multi_meta_aln.get_parsed_alignments()
concat_tuple = multi_meta_aln.get_concatenated(parsed_alns)
concatenated_alignments = concat_tuple[0]
concatenated_partitions = concat_tuple[1]
```

Removing taxa from alignments is very easy:

```
spp_to_remove = ["taxon1", "taxon2", "taxon3"]
reduced_alns = multi_meta_aln.remove_taxa(spp_to_remove)
```

To print to file or convert among file formats use one of the `.print_format(parsed_alignment)` methods called with a parsed dictionary as an argument. These methods include `.print_fasta()`, `.print_nexus()`, `.print_nexus_int()`, `print_phylip()`, and `.print_phylip_int()`. They return an apporpriately formatted string.

```
for alignment in concatenated_alignments:
    nex_int_string = meta_aln.print_nexus_int(alignment)
    print(nex_int_string)
```

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

amas 19
amas.AMAS 20

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

- amas.AMAS.Alignment 23
  - amas.AMAS.AminoAcidAlignment 45
  - amas.AMAS.DNAAlignment 48
- amas.AMAS.FileHandler 53
- amas.AMAS.FileParser 55
- amas.AMAS.MetaAlignment 64
- amas.AMAS.ParsedArgs 115

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

amas.AMAS.Alignment 23
amas.AMAS.AminoAcidAlignment 45
amas.AMAS.DNAAlignment 48
amas.AMAS.FileHandler 53
amas.AMAS.FileParser 55
amas.AMAS.MetaAlignment 64
amas.AMAS.ParsedArgs 115

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

amas/__init__.py 127
amas/AMAS.py 127

Chapter 6

Namespace Documentation

6.1 amas Namespace Reference

Namespaces

- namespace AMAS

Variables

- str __author__ = 'Marek Borowiec'
- str __email__ = 'petiolus@gmail.com'
- str __version__ = '1.02'
- __all__ = dir()

6.1.1 Variable Documentation

6.1.1.1 __all__

```
amas.__all__ = dir()  [private]
```

Definition at line 6 of file __init__.py.

6.1.1.2 __author__

```
str amas.__author__ = 'Marek Borowiec'  [private]
```

Definition at line 3 of file __init__.py.

6.1.1.3 __email__

```
str amas.__email__ = 'petiolus@gmail.com'  [private]
```

Definition at line 4 of file __init__.py.

6.1.1.4 __version__

```
str amas.__version__ = '1.02'  [private]
```

Definition at line 5 of file __init__.py.

6.2 amas.AMAS Namespace Reference

Classes

- class Alignment
- class AminoAcidAlignment
- class DNAAlignment
- class FileHandler
- class FileParser
- class MetaAlignment
- class ParsedArgs

Functions

- proportion (x)
- main ()
- run ()

6.2.1 Detailed Description

```
This stand-alone program allows manipulations of multiple sequence
alignments. It supports sequential FASTA, PHYLIP, NEXUS, and interleaved PHYLIP
and NEXUS formats for DNA and aino acid sequences. It can print summary statistics,
convert among formats, and concatenate alignments.

Current statistics include the number of taxa, alignment length, total number
of matrix cells, overall number of undetermined characters, percent of missing
data, AT and GC contents (for DNA alignments), number and proportion of
variable sites, number and proportion of parsimony informative sites,
and counts of all characters present in the relevant (nucleotide or amino acid) alphabet.
```

6.2.2 Function Documentation

6.2.2.1 main()

```
amas.AMAS.main ( )
```

Definition at line 2361 of file AMAS.py.

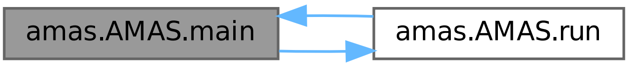
```
02361 def main():
02362
02363     # initialize parsed arguments and meta alignment objects
02364     kwargs = run()
02365     meta_aln = MetaAlignment(**kwargs)
02366
02367     if meta_aln.command == "summary":
02368         meta_aln.write_summaries(kwargs["summary_out"])
02369     if meta_aln.by_taxon_summary:
02370         print("Printing taxon summaries")
02371         meta_aln.write_taxa_summaries()
```

```
    if meta_aln.command == "convert":
        meta_aln.write_out("convert", kwargs["out_format"])
    if meta_aln.command == "concat":
        meta_aln.write_out("concat", kwargs["out_format"])
        meta_aln.write_partitions(kwargs["concat_part"], kwargs["part_format"], kwargs["data_type"], kwargs["codons"])
    if meta_aln.command == "replicate":
        meta_aln.write_out("replicate", kwargs["out_format"])
    if meta_aln.command == "split":
        meta_aln.write_out("split", kwargs["out_format"])
    if meta_aln.command == "remove":
        meta_aln.write_out("remove", kwargs["out_format"])
    if meta_aln.command == "translate":
        meta_aln.write_out("translate", kwargs["out_format"])
    if meta_aln.command == "trim":
        meta_aln.write_out("trim", kwargs["out_format"])

    if meta_aln.command == "metapartitions":
        # `metapartitions` is essentially `split` + `concat`. Currently you can't set an out_format:
        # it's automatically set to match the in_format because the intermediate `split` outputs become
        # the 'new' in_files for the `concat` operation, and then calling either:
        #  -> AminoAcidAlignment(Alignment.__init__(self, in_file, in_format, data_type))
        #  -> DNAAlignment(Alignment.__init__(self, in_file, in_format, data_type))
        # through MetaAlignment.get_alignment_object(alignment, self.in_format, self.data_type)
        meta_aln.write_out("metapartitions", kwargs["in_format"])
        meta_aln.write_partitions(kwargs["concat_part"], kwargs["part_format"], kwargs["data_type"], "none")

        # meta_aln.write_out("translate", kwargs["out_format"])

```

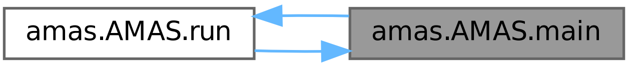
References amas.AMAS.run().

Referenced by amas.AMAS.run().

Here is the call graph for this function:



Here is the caller graph for this function:



6.2.2.2 proportion()

```
amas.AMAS.proportion (
                x )
```

Definition at line 43 of file AMAS.py.

```
00043 def proportion(x):
00044     # needed to prevent input of invalid floats in trim mode
00045     x = float(x)
00046     if x < 0.0 or x > 1.0:
00047         raise argparse.ArgumentTypeError("%r not in range [0.0, 1.0]" % (x,))
00048     return x
00049
```

6.2.2.3 run()

`amas.AMAS.run ( )`

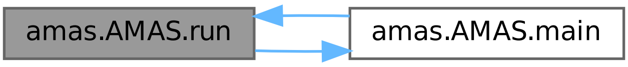
Definition at line 2400 of file AMAS.py.

```
02400 def run():
02401
02402     # initialize parsed arguments
02403     config = ParsedArgs()
02404     # get arguments
02405     config_dict = config.get_args_dict()
02406     return config_dict
02407
```

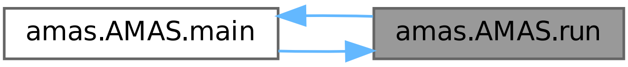
References amas.AMAS.main().

Referenced by amas.AMAS.main().

Here is the call graph for this function:



Here is the caller graph for this function:

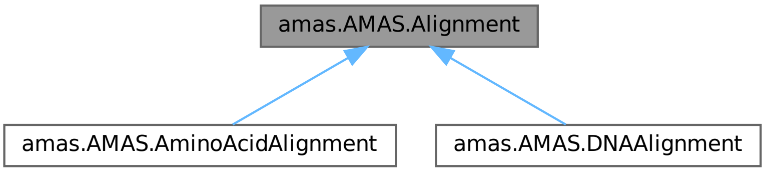


Chapter 7

Class Documentation

7.1 amas.AMAS.Alignment Class Reference

Inheritance diagram for amas.AMAS.Alignment:


amas.AMAS.Alignment
amas.AMAS.AminoAcidAlignment
amas.AMAS.DNAAlignment

Public Member Functions

- __init__ (self, in_file, in_format, data_type)
- __str__ (self)
- get_aln_input (self)
- get_parsed_aln (self)
- summarize_alignment (self)
- summarize_alignment_by_taxa (self)
- get_char_summary (self)
- get_taxon_char_summary (self)
- append_count (self, char_dict)
- matrix_creator (self)
- get_column (self, i)
- all_same (self, site)
- get_sites_no_missing_ambiguous (self)
- get_site_no_missing_ambiguous (self, column)
- replace_missing (self, column)
- get_trim_selection (self, trim_fraction, parsimony_check)
- get_variable (self)
- get_parsimony_informative (self)

- get_prop_variable (self)
- get_prop_parsimony (self)
- get_name (self)
- get_taxa_no (self)
- get_alignment_length (self)
- get_matrix_cells (self)
- get_missing (self)
- get_missing_percent (self)
- get_missing_from_parsed (self)
- get_missing_from_seq (self, seq)
- get_missing_percent_from_seq (self, seq)
- get_counts (self)
- get_counts_from_parsed (self)
- get_counts_from_seq (self, seq)
- check_data_type (self)

Public Attributes

- in_file
- in_format
- data_type
- parsed_aln
- length
- matrix
- no_missing_ambiguous
- variable_sites
- prop_variable
- parsimony_informative
- prop_parsimony
- missing_records
- missing
- check

Static Public Attributes

- all_matrix_cells

7.1.1 Detailed Description

```
Base class: Gets in parsed sequences as input and summarizes their stats.
Based on the data type, the subclasses AminoAcidAlignment & DNAAlignment define the attributes:
`alphabet`, `missing_ambiguous_chars`, `missing_chars`, `non_alphabet`
```

Definition at line 805 of file AMAS.py.

7.1.2 Constructor & Destructor Documentation

7.1.2.1 __init__()

```
amas.AMAS.Alignment.__init__ (
                self,
                in_file,
                in_format,
                data_type )
```

Definition at line 811 of file AMAS.py.

```
00811     def __init__(self, in_file, in_format, data_type):
00812         # initialize alignment class with parsed records and alignment name as arguments,
00813         # create empty lists for list of sequences, sites without
00814         # ambiguous or missing characters, and initialize variable for the number
00815         # of parsimony informative sites
00816         self.in_file = in_file
00817         self.in_format = in_format
00818         self.data_type = data_type
00819 
00820         self.parsed_aln = self.get_parsed_aln()
00821 
```

7.1.3 Member Function Documentation

7.1.3.1 __str__()

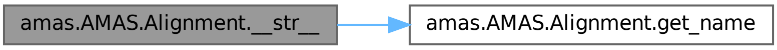
```
amas.AMAS.Alignment.__str__ (
                self )
```

Definition at line 822 of file AMAS.py.

```
00822     def __str__(self):
00823         #  purpose of override? (originally returned method object)
00824         return self.get_name()
00825 
```

References amas.AMAS.Alignment.get_name().

Here is the call graph for this function:



7.1.3.2 all_same()

```
amas.AMAS.Alignment.all_same (
                self,
                site )
```

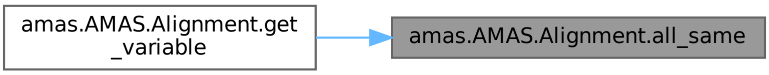
Definition at line 937 of file AMAS.py.

```
00937     def all_same(self, site):
00938         # check if all elements of a site are the same
00939         return not site or site.count(site[0]) == len(site)
```

```
00940
```

Referenced by amas.AMAS.Alignment.get_variable().

Here is the caller graph for this function:


amas.AMAS.Alignment.get_variable
amas.AMAS.Alignment.all_same

7.1.3.3 append_count()

```
amas.AMAS.Alignment.append_count (
                self,
                char_dict )
```

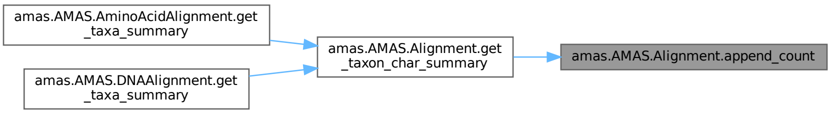
Definition at line 919 of file AMAS.py.

```
00919     def append_count(self, char_dict):
00920         count_list = []
00921         for char in self.alphabet:
00922             if char in char_dict.keys():
00923                 count_list.append(char_dict[char])
00924             else:
00925                 count_list.append(0)
00926         return count_list
00927
```

References amas.AMAS.AminoAcidAlignment.alphabet, and amas.AMAS.DNAAlignment.alphabet.

Referenced by amas.AMAS.Alignment.get_taxon_char_summary().

Here is the caller graph for this function:


amas.AMAS.AminoAcidAlignment.get_taxa_summary
amas.AMAS.DNAAlignment.get_taxa_summary
amas.AMAS.Alignment.get_taxon_char_summary
amas.AMAS.Alignment.append_count

7.1.3.4 check_data_type()

```
amas.AMAS.Alignment.check_data_type (
                self )
```

Definition at line 1074 of file AMAS.py.

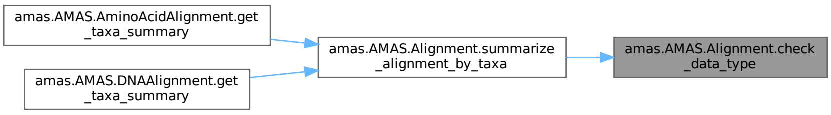
```
01074     def check_data_type(self):
01075         # check if the data type is correct; only one seq to save on computation
01076         seq = next(iter(self.parsed_aln.values()))
01077         self.check = any(char in self.non_alphabet for char in seq)
01078         if self.check is True:
```

```
01079             print(
01080                 "WARNING: found non-" + self.data_type + " characters. "
01081                 "Are you sure you specified the right data type?"
01082             )
01083 
01084 
```

References amas.AMAS.Alignment.parsed_aln.

Referenced by amas.AMAS.Alignment.summarize_alignment_by_taxa().

Here is the caller graph for this function:


amas.AMAS.AminoAcidAlignment.get
_taxa_summary
amas.AMAS.DNAAlignment.get
_taxa_summary
amas.AMAS.Alignment.summarize
_alignment_by_taxa
amas.AMAS.Alignment.check
_data_type

7.1.3.5 get_alignment_length()

amas.AMAS.Alignment.get_alignment_length (
self)

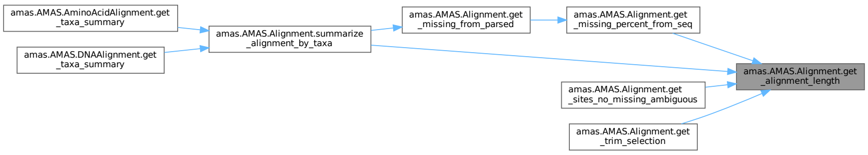
Definition at line 1011 of file AMAS.py.

```
01011     def get_alignment_length(self):
01012         # get alignment length by just checking the first seq length
01013         # this assumes that all sequences are of equal length
01014         return len(next(iter(self.parsed_aln.values())))
01015 
```

References amas.AMAS.Alignment.parsed_aln.

Referenced by amas.AMAS.Alignment.get_missing_percent_from_seq(), amas.AMAS.Alignment.get_sites_no_missing_ambiguous()
amas.AMAS.Alignment.get_trim_selection(), and amas.AMAS.Alignment.summarize_alignment_by_taxa().

Here is the caller graph for this function:


amas.AMAS.AminoAcidAlignment.get
_taxa_summary
amas.AMAS.DNAAlignment.get
_taxa_summary
amas.AMAS.Alignment.summarize
_alignment_by_taxa
amas.AMAS.Alignment.get
_missing_from_parsed
amas.AMAS.Alignment.get
_missing_percent_from_seq
amas.AMAS.Alignment.get
_alignment_length
amas.AMAS.Alignment.get
_sites_no_missing_ambiguous
amas.AMAS.Alignment.get
_trim_selection

7.1.3.6 get_aln_input()

```
amas.AMAS.Alignment.get_aln_input (
                self )
```

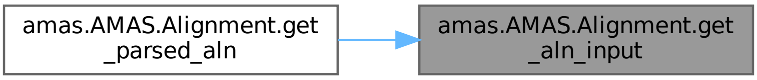
Definition at line 826 of file AMAS.py.

```
00826     def get_aln_input(self):
00827         # open and parse input file
00828         aln_input = FileParser(self.in_file)
00829         return aln_input
00830
```

References amas.AMAS.FileHandler.in_file, amas.AMAS.FileParser.in_file, and amas.AMAS.Alignment.in_file.

Referenced by amas.AMAS.Alignment.get_parsed_aln().

Here is the caller graph for this function:


amas.AMAS.Alignment.get _parsed_aln
amas.AMAS.Alignment.get _aln_input

7.1.3.7 get_char_summary()

```
amas.AMAS.Alignment.get_char_summary (
                self )
```

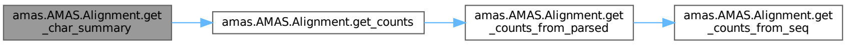
Definition at line 899 of file AMAS.py.

```
00899     def get_char_summary(self):
00900         # get summary of frequencies for all characters
00901         characters = []
00902         counts = []
00903         add_to_chars = characters.append
00904         add_to_counts = counts.append
00905         char_count_dicts = self.get_counts()
00906         for char in self.alphabet:
00907             add_to_chars(char)
00908             if char in char_count_dicts.keys():
00909                 add_to_counts(str(char_count_dicts[char]))
00910             else:
00911                 add_to_counts("0")
00912         return characters, counts
00913
```

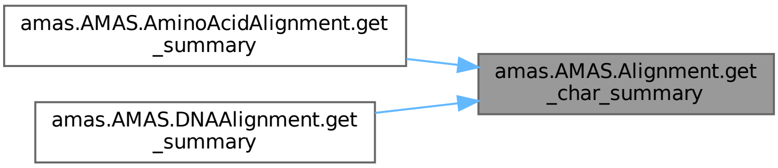
References amas.AMAS.AminoAcidAlignment.alphabet, amas.AMAS.DNAAlignment.alphabet, and amas.AMAS.Alignment.get_coun

Referenced by amas.AMAS.AminoAcidAlignment.get_summary(), and amas.AMAS.DNAAlignment.get_summary().

Here is the call graph for this function:


amas.AMAS.Alignment.get _char_summary
amas.AMAS.Alignment.get_counts
amas.AMAS.Alignment.get _counts_from_parsed
amas.AMAS.Alignment.get _counts_from_seq

Here is the caller graph for this function:


amas.AMAS.AminoAcidAlignment.get
_summary
amas.AMAS.DNAAlignment.get
_summary
amas.AMAS.Alignment.get
_char_summary

7.1.3.8 get_column()

```
amas.AMAS.Alignment.get_column (
                self,
                i )
```

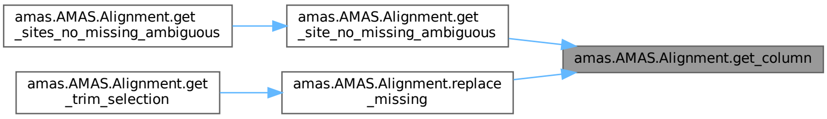
Definition at line 933 of file AMAS.py.

```
00933     def get_column(self, i):
00934         # get site from the character matrix
00935         return [row[i] for row in self.matrix]
00936
```

References amas.AMAS.Alignment.matrix.

Referenced by amas.AMAS.Alignment.get_site_no_missing_ambiguous(), and amas.AMAS.Alignment.replace_missing().

Here is the caller graph for this function:


amas.AMAS.Alignment.get
_sites_no_missing_ambiguous
amas.AMAS.Alignment.get
_site_no_missing_ambiguous
amas.AMAS.Alignment.get_column
amas.AMAS.Alignment.get
_trim_selection
amas.AMAS.Alignment.replace
_missing

7.1.3.9 get_counts()

```
amas.AMAS.Alignment.get_counts (
                self )
```

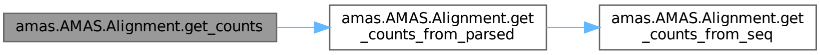
Definition at line 1052 of file AMAS.py.

```
01052     def get_counts(self):
01053         # get counts of each character in the used alphabet for all sequences
01054         counters = [Counter(chars) for taxon, chars in self.get_counts_from_parsed()]
01055         all_counts = sum(counters, Counter())
01056         counts_dict = dict(all_counts)
01057         return counts_dict
01058
```

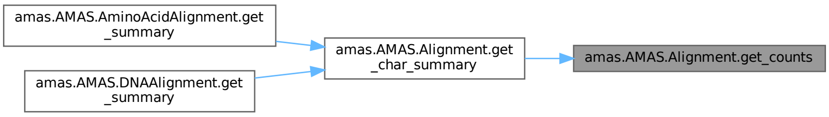
References amas.AMAS.Alignment.get_counts_from_parsed().

Referenced by amas.AMAS.Alignment.get_char_summary().

Here is the call graph for this function:


amas.AMAS.Alignment.get_counts
amas.AMAS.Alignment.get
_counts_from_parsed
amas.AMAS.Alignment.get
_counts_from_seq

Here is the caller graph for this function:


amas.AMAS.AminoAcidAlignment.get
_summary
amas.AMAS.DNAAlignment.get
_summary
amas.AMAS.Alignment.get
_char_summary
amas.AMAS.Alignment.get_counts

7.1.3.10 get_counts_from_parsed()

```
amas.AMAS.Alignment.get_counts_from_parsed (
                self )
```

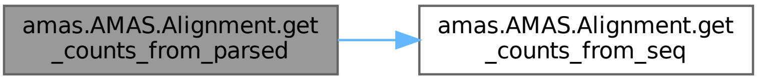
Definition at line 1059 of file AMAS.py.

```
    def get_counts_from_parsed(self):
        # get counts of all characters from parsed alignment
        # return a list of tuples with taxon name and counts
        return sorted(
            [
                (taxon, self.get_counts_from_seq(seq))
                for taxon, seq in self.parsed_aln.items()
            ]
        )

```

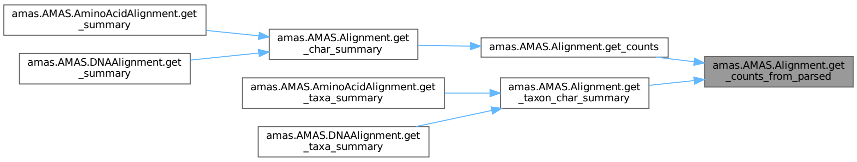
References amas.AMAS.Alignment.get_counts_from_seq(), and amas.AMAS.Alignment.parsed_aln.

Referenced by amas.AMAS.Alignment.get_counts(), and amas.AMAS.Alignment.get_taxon_char_summary().

Here is the call graph for this function:


amas.AMAS.Alignment.get
_counts_from_parsed
amas.AMAS.Alignment.get
_counts_from_seq

Here is the caller graph for this function:


amas.AMAS.AminoAcidAlignment.get_summary
amas.AMAS.DNAAlignment.get_summary
amas.AMAS.Alignment.get_char_summary
amas.AMAS.AminoAcidAlignment.get_taxa_summary
amas.AMAS.DNAAlignment.get_taxa_summary
amas.AMAS.Alignment.get_counts
amas.AMAS.Alignment.get_taxon_char_summary
amas.AMAS.Alignment.get_counts_from_parsed

7.1.3.11 get_counts_from_seq()

```
amas.AMAS.Alignment.get_counts_from_seq (
                self,
                seq )
```

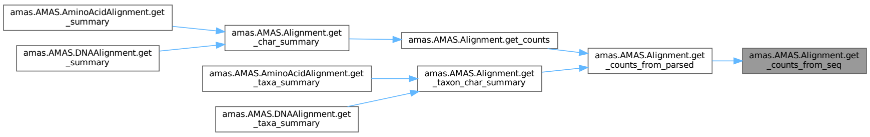
Definition at line 1069 of file AMAS.py.

```
01069     def get_counts_from_seq(self, seq):
01070         # get all alphabet chars count for individual sequence
01071         char_counts = {char : seq.count(char) for char in self.alphabet}
01072         return char_counts
01073
```

References amas.AMAS.AminoAcidAlignment.alphabet, and amas.AMAS.DNAAlignment.alphabet.

Referenced by amas.AMAS.Alignment.get_counts_from_parsed().

Here is the caller graph for this function:


amas.AMAS.AminoAcidAlignment.get_summary
amas.AMAS.DNAAlignment.get_summary
amas.AMAS.Alignment.get_char_summary
amas.AMAS.AminoAcidAlignment.get_taxa_summary
amas.AMAS.DNAAlignment.get_taxa_summary
amas.AMAS.Alignment.get_counts
amas.AMAS.Alignment.get_taxon_char_summary
amas.AMAS.Alignment.get_counts_from_parsed
amas.AMAS.Alignment.get_counts_from_seq

7.1.3.12 get_matrix_cells()

```
amas.AMAS.Alignment.get_matrix_cells (
                self )
```

Definition at line 1016 of file AMAS.py.

```
01016     def get_matrix_cells(self):
```

7.1.3.13 get_missing()

```
amas.AMAS.Alignment.get_missing (
                self )
```

Definition at line 1021 of file AMAS.py.

```
01021     def get_missing(self):
01022         # count missing characters from the list of missing for all sequences
01023         self.missing = sum(count for taxon, count, percent in self.missing_records)
01024         return self.missing
01025
```

7.1.3.14 get_missing_from_parsed()

```
amas.AMAS.Alignment.get_missing_from_parsed (
              self )
```

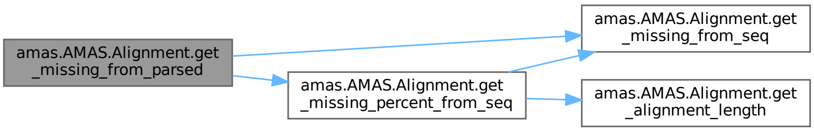
Definition at line 1031 of file AMAS.py.

```
01031     def get_missing_from_parsed(self):
01032         # get missing count and percent from parsed alignment
01033         # return a list of tuples with taxon name, count, and percent missing
01034         self.missing_records = sorted(
01035             [
01036                 (taxon, self.get_missing_from_seq(seq), self.get_missing_percent_from_seq(seq))
01037                 for taxon, seq in self.parsed_aln.items()
01038             ]
01039         )
01040         return self.missing_records
01041 
```

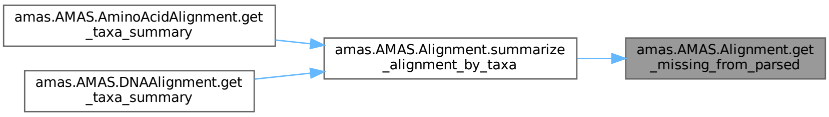
References amas.AMAS.Alignment.get_missing_from_seq(), amas.AMAS.Alignment.get_missing_percent_from_seq(), amas.AMAS.Alignment.missing_records, and amas.AMAS.Alignment.parsed_aln.

Referenced by amas.AMAS.Alignment.summarize_alignment_by_taxa().

Here is the call graph for this function:


amas.AMAS.Alignment.get _missing_from_parsed
amas.AMAS.Alignment.get _missing_from_seq
amas.AMAS.Alignment.get _missing_percent_from_seq
amas.AMAS.Alignment.get _alignment_length

Here is the caller graph for this function:


amas.AMAS.AminoAcidAlignment.get _taxa_summary
amas.AMAS.DNAAlignment.get _taxa_summary
amas.AMAS.Alignment.summarize _alignment_by_taxa
amas.AMAS.Alignment.get _missing_from_parsed

7.1.3.15 get_missing_from_seq()

```
amas.AMAS.Alignment.get_missing_from_seq (
              self,
              seq )
```

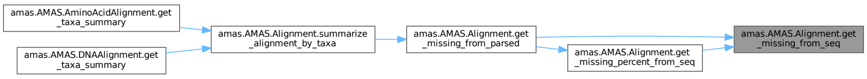
Definition at line 1042 of file AMAS.py.

```
01042     def get_missing_from_seq(self, seq):
01043         # count missing characters for individual sequence
01044         missing_count = sum(seq.count(char) for char in self.missing_chars)
01045         return missing_count
01046 
```

References amas.AMAS.AminoAcidAlignment.missing_chars, and amas.AMAS.DNAAlignment.missing_chars.

Referenced by amas.AMAS.Alignment.get_missing_from_parsed(), and amas.AMAS.Alignment.get_missing_percent_from_seq().

Here is the caller graph for this function:


amas.AMAS.AminoAcidAlignment.get
_taxa_summary
amas.AMAS.DNAAlignment.get
_taxa_summary
amas.AMAS.Alignment.summarize
_alignment_by_taxa
amas.AMAS.Alignment.get
_missing_from_parsed
amas.AMAS.Alignment.get
_missing_percent_from_seq
amas.AMAS.Alignment.get
_missing_from_seq

7.1.3.16 get_missing_percent()

```
amas.AMAS.Alignment.get_missing_percent (
                self )
```

Definition at line 1026 of file AMAS.py.

```
01026     def get_missing_percent(self):
01027         # get missing percent
01028         missing_percent = round((self.missing / self.all_matrix_cells * 100), 3)
01029         return missing_percent
01030
```

References amas.AMAS.Alignment.all_matrix_cells, and amas.AMAS.Alignment.missing.

7.1.3.17 get_missing_percent_from_seq()

```
amas.AMAS.Alignment.get_missing_percent_from_seq (
                self,
                seq )
```

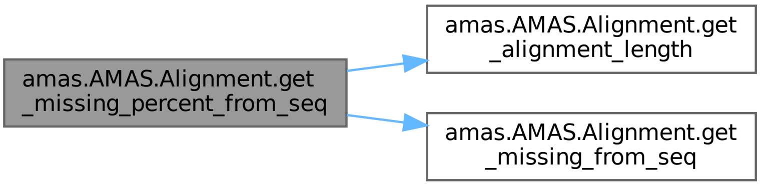
Definition at line 1047 of file AMAS.py.

```
01047     def get_missing_percent_from_seq(self, seq):
01048         # get missing percent from individual sequence
01049         missing_seq_percent = round((self.get_missing_from_seq(seq) / self.get_alignment_length() *
      100), 3)
01050         return missing_seq_percent
01051
```

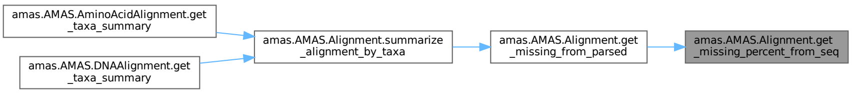
References amas.AMAS.Alignment.get_alignment_length(), and amas.AMAS.Alignment.get_missing_from_seq().

Referenced by amas.AMAS.Alignment.get_missing_from_parsed().

Here is the call graph for this function:


amas.AMAS.Alignment.get
_missing_percent_from_seq
amas.AMAS.Alignment.get
_alignment_length
amas.AMAS.Alignment.get
_missing_from_seq

Here is the caller graph for this function:


amas.AMAS.AminoAcidAlignment.get
_taxa_summary
amas.AMAS.DNAAlignment.get
_taxa_summary
amas.AMAS.Alignment.summarize
_alignment_by_taxa
amas.AMAS.Alignment.get
_missing_from_parsed
amas.AMAS.Alignment.get
_missing_percent_from_seq

7.1.3.18 get_name()

```
amas.AMAS.Alignment.get_name (
                self )
```

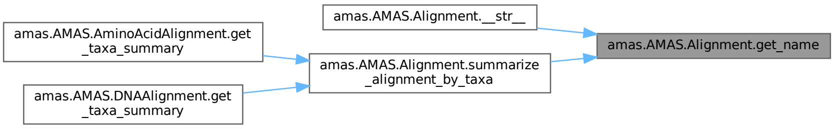
Definition at line 1002 of file AMAS.py.

```
01002     def get_name(self):
01003         # get input file name
01004         in_filename = path.basename(self.in_file)
01005         return in_filename
01006
```

References amas.AMAS.FileHandler.in_file, amas.AMAS.FileParser.in_file, and amas.AMAS.Alignment.in_file.

Referenced by amas.AMAS.Alignment.__str__(), and amas.AMAS.Alignment.summarize_alignment_by_taxa().

Here is the caller graph for this function:


amas.AMAS.Alignment.__str__
amas.AMAS.AminoAcidAlignment.get
_taxa_summary
amas.AMAS.Alignment.summarize
_alignment_by_taxa
amas.AMAS.DNAAlignment.get
_taxa_summary
amas.AMAS.Alignment.get_name

7.1.3.19 get_parsed_aln()

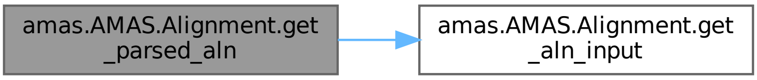
```
amas.AMAS.Alignment.get_parsed_aln (
                self )
```

Definition at line 831 of file AMAS.py.

```
00831     def get_parsed_aln(self):
00832         # parse according to the given format
00833         aln_input = self.get_aln_input()
00834         if self.in_format == "fasta":
00835             parsed_aln = aln_input.fasta_parse()
00836         elif self.in_format == "phylip":
00837             parsed_aln = aln_input.phylip_parse()
00838         elif self.in_format == "phylip-int":
00839             parsed_aln = aln_input.phylip_interleaved_parse()
00840         elif self.in_format == "nexus":
00841             parsed_aln = aln_input.nexus_parse()
00842         elif self.in_format == "nexus-int":
00843             parsed_aln = aln_input.nexus_interleaved_parse()
00844
00845         return parsed_aln
00846
```

References amas.AMAS.Alignment.get_aln_input(), amas.AMAS.Alignment.in_format, and amas.AMAS.MetaAlignment.in_format.

Here is the call graph for this function:



7.1.3.20 get_parsimony_informative()

```
amas.AMAS.Alignment.get_parsimony_informative (
                self )
```

Definition at line 978 of file AMAS.py.

```
00978     def get_parsimony_informative(self):
00979         # if the count for a unique character in a site is at least two,
00980         # and there are at least two such characters in a site without missing
00981         # or ambiguous characters, consider it parsimony informative
00982         parsimony_informative = 0
00983         for site in self.no_missing_ambiguous:
00984             unique_chars = set(site)
00985             pattern = [base for base in unique_chars if site.count(base) >= 2]
00986             no_patterns = len(pattern)
00987 
00988             if no_patterns >= 2:
00989                 parsimony_informative += 1
00990         return parsimony_informative
00991 
```

References amas.AMAS.Alignment.no_missing_ambiguous.

7.1.3.21 get_prop_parsimony()

```
amas.AMAS.Alignment.get_prop_parsimony (
                self )
```

Definition at line 997 of file AMAS.py.

```
00997     def get_prop_parsimony(self):
00998         # get proportion of parsimony informative sites to all sites
00999         prop_parsimony = self.parsimony_informative / int(self.length)
01000         return round(prop_parsimony, 3)
01001 
```

References amas.AMAS.Alignment.length, and amas.AMAS.Alignment.parsimony_informative.

7.1.3.22 get_prop_variable()

```
amas.AMAS.Alignment.get_prop_variable (
                self )
```

Definition at line 992 of file AMAS.py.

```
00992     def get_prop_variable(self):
00993         # get proportion of variable sites to all sites
00994         prop_variable = self.variable_sites / int(self.length)
00995         return round(prop_variable, 3)
00996 
```

References amas.AMAS.Alignment.length, and amas.AMAS.Alignment.variable_sites.

7.1.3.23 get_site_no_missing_ambiguous()

```
amas.AMAS.Alignment.get_site_no_missing_ambiguous (
                self,
                column )
```

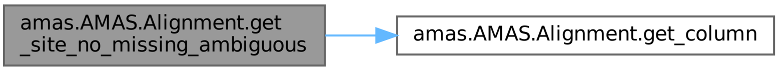
Definition at line 946 of file AMAS.py.

```
00946     def get_site_no_missing_ambiguous(self, column):
00947         site = self.get_column(column)
00948         return [char for char in site if char not in self.missing_ambiguous_chars]
00949
```

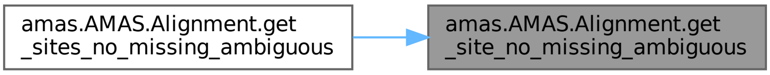
References amas.AMAS.Alignment.get_column(), amas.AMAS.AminoAcidAlignment.missing_ambiguous_chars, and amas.AMAS.DNAAlignment.missing_ambiguous_chars.

Referenced by amas.AMAS.Alignment.get_sites_no_missing_ambiguous().

Here is the call graph for this function:


amas.AMAS.Alignment.get
_site_no_missing_ambiguous
amas.AMAS.Alignment.get_column

Here is the caller graph for this function:


amas.AMAS.Alignment.get
_sites_no_missing_ambiguous
amas.AMAS.Alignment.get
_site_no_missing_ambiguous

7.1.3.24 get_sites_no_missing_ambiguous()

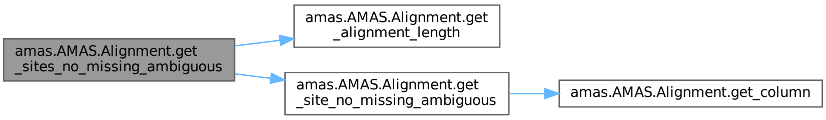
```
amas.AMAS.Alignment.get_sites_no_missing_ambiguous (
                self )
```

Definition at line 941 of file AMAS.py.

```
00941     def get_sites_no_missing_ambiguous(self):
00942         # get each site without missing or ambiguous characters
00943         no_missing_ambiguous_sites = [self.get_site_no_missing_ambiguous(column) for column in
      range(self.get_alignment_length())]
00944         return no_missing_ambiguous_sites
00945
```

References amas.AMAS.Alignment.get_alignment_length(), and amas.AMAS.Alignment.get_site_no_missing_ambiguous().

Here is the call graph for this function:


amas.AMAS.Alignment.get
_sites_no_missing_ambiguous
amas.AMAS.Alignment.get
_alignment_length
amas.AMAS.Alignment.get
_site_no_missing_ambiguous
amas.AMAS.Alignment.get_column

7.1.3.25 get_taxa_no()

```
amas.AMAS.Alignment.get_taxa_no (
                self )
```

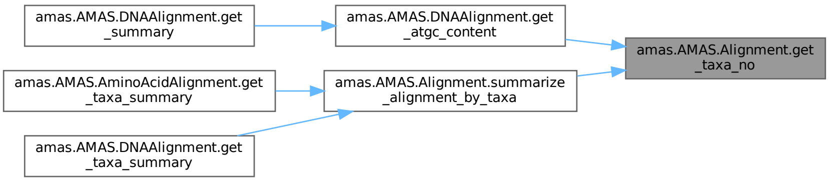
Definition at line 1007 of file AMAS.py.

```
01007     def get_taxa_no(self):
01008         # get number of taxa
01009         return len(self.parsed_aln.values())
01010
```

References amas.AMAS.Alignment.parsed_aln.

Referenced by amas.AMAS.DNAAlignment.get_atgc_content(), and amas.AMAS.Alignment.summarize_alignment_by_taxa().

Here is the caller graph for this function:


amas.AMAS.DNAAlignment.get_summary
amas.AMAS.DNAAlignment.get_atgc_content
amas.AMAS.Alignment.get_taxa_no
amas.AMAS.AminoAcidAlignment.get_taxa_summary
amas.AMAS.Alignment.summarize_alignment_by_taxa
amas.AMAS.DNAAlignment.get_taxa_summary

7.1.3.26 get_taxon_char_summary()

```
amas.AMAS.Alignment.get_taxon_char_summary (
                self )
```

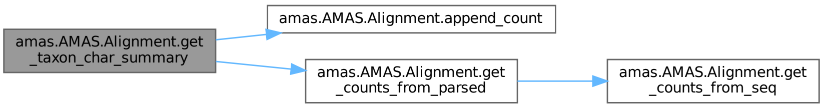
Definition at line 914 of file AMAS.py.

```
00914     def get_taxon_char_summary(self):
00915         # get summary of frequencies for all characters
00916         records = (self.append_count(char_dict) for taxon, char_dict in self.get_counts_from_parsed())
00917         return records
00918
```

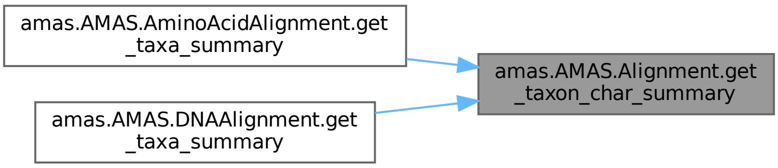
References amas.AMAS.Alignment.append_count(), and amas.AMAS.Alignment.get_counts_from_parsed().

Referenced by amas.AMAS.AminoAcidAlignment.get_taxa_summary(), and amas.AMAS.DNAAlignment.get_taxa_summary().

Here is the call graph for this function:


amas.AMAS.Alignment.get_taxon_char_summary
amas.AMAS.Alignment.append_count
amas.AMAS.Alignment.get_counts_from_parsed
amas.AMAS.Alignment.get_counts_from_seq

Here is the caller graph for this function:


amas.AMAS.AminoAcidAlignment.get_taxa_summary
amas.AMAS.DNAAlignment.get_taxa_summary
amas.AMAS.Alignment.get_taxon_char_summary

7.1.3.27 get_trim_selection()

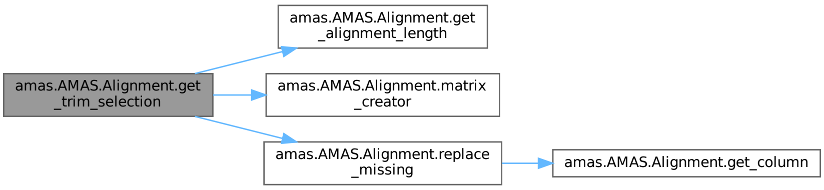
```
amas.AMAS.Alignment.get_trim_selection (
                self,
                trim_fraction,
                parsimony_check )
```

Definition at line 953 of file AMAS.py.

```
00953     def get_trim_selection(self, trim_fraction, parsimony_check):
00954         # this checks each column of alignment for minimum occupancy
00955         self.matrix = self.matrix_creator()
00956         trim_vector = []
00957         for column in range(self.get_alignment_length()):
00958             site = self.replace_missing(column)
00959             occ = (len(site) - site.count("-")) / len(site)
00960             if parsimony_check:
00961                 unique_chars = set(site)
00962                 try:
00963                     unique_chars.remove("-")
00964                 except KeyError:
00965                     pass # this occurs if we have no missing data
00966                 pattern = [base for base in unique_chars if site.count(base) >= 2]
00967                 trim_vector.append(len(pattern) >= 2 and occ >= trim_fraction)
00968             else:
00969                 trim_vector.append(occ >= trim_fraction)
00970         return trim_vector
00971 
```

References amas.AMAS.Alignment.get_alignment_length(), amas.AMAS.Alignment.matrix, amas.AMAS.Alignment.matrix_creator(), and amas.AMAS.Alignment.replace_missing().

Here is the call graph for this function:


amas.AMAS.Alignment.get_trim_selection
amas.AMAS.Alignment.get_alignment_length
amas.AMAS.Alignment.matrix_creator
amas.AMAS.Alignment.replace_missing
amas.AMAS.Alignment.get_column

7.1.3.28 get_variable()

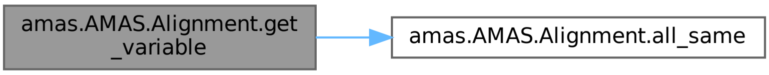
```
amas.AMAS.Alignment.get_variable (
              self )
```

Definition at line 972 of file AMAS.py.

```
00972     def get_variable(self):
00973         # if all elements of a site without missing or ambiguous characters
00974         # are not the same, consider it variable
00975         variable = len([site for site in self.no_missing_ambiguous if not self.all_same(site)])
00976         return variable
00977
```

References amas.AMAS.Alignment.all_same(), and amas.AMAS.Alignment.no_missing_ambiguous.

Here is the call graph for this function:


amas.AMAS.Alignment.get
_variable
amas.AMAS.Alignment.all_same

7.1.3.29 matrix_creator()

```
amas.AMAS.Alignment.matrix_creator (
              self )
```

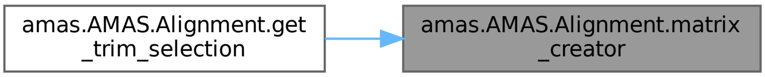
Definition at line 928 of file AMAS.py.

```
00928     def matrix_creator(self):
00929         # decompose character matrix into a two-dimensional list
00930         matrix = [list(sequence) for sequence in self.parsed_aln.values()]
00931         return matrix
00932
```

References amas.AMAS.Alignment.parsed_aln.

Referenced by amas.AMAS.Alignment.get_trim_selection().

Here is the caller graph for this function:


amas.AMAS.Alignment.get
_trim_selection
amas.AMAS.Alignment.matrix
_creator

7.1.3.30 replace_missing()

```
amas.AMAS.Alignment.replace_missing (
                self,
                column )
```

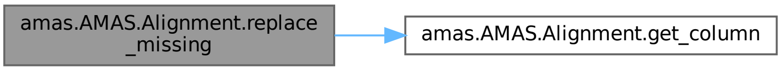
Definition at line 950 of file AMAS.py.

```
    def replace_missing(self, column):
        return ["-" if x in self.missing_chars else x for x in self.get_column(column)]

```

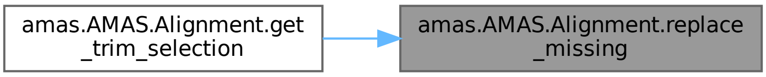
References amas.AMAS.Alignment.get_column(), amas.AMAS.AminoAcidAlignment.missing_chars, and amas.AMAS.DNAAlignment.missing_chars.

Referenced by amas.AMAS.Alignment.get_trim_selection().

Here is the call graph for this function:



Here is the caller graph for this function:



7.1.3.31 summarize_alignment()

```
amas.AMAS.Alignment.summarize_alignment (
                self )
```

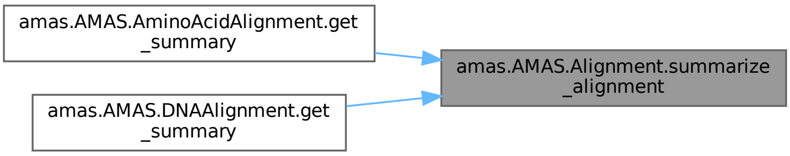
Definition at line 847 of file AMAS.py.

```
    def summarize_alignment(self):
        # call methods to create sequences list, matrix, sites without ambiguous or
        # missing characters; get and summarize alignment statistics
        summary = []
        self.length = str(self.get_alignment_length())
        self.matrix = self.matrix_creator()
        self.no_missing_ambiguous = self.get_sites_no_missing_ambiguous()
        self.variable_sites = self.get_variable()
        self.prop_variable = self.get_prop_variable()
        self.parsimony_informative = self.get_parsimony_informative()
        self.prop_parsimony = self.get_prop_parsimony()
        self.missing_records = self.get_missing_from_parsed()
        name = str(self.get_name())
        taxa_no = str(self.get_taxa_no())
        cells = str(self.get_matrix_cells())
        missing = str(self.get_missing())
        missing_percent = str(self.get_missing_percent())
```

```
00864         self.check_data_type()
00865         summary = [
00866             name,
00867             taxa_no,
00868             self.length,
00869             cells,
00870             missing,
00871             missing_percent,
00872             str(self.variable_sites),
00873             str(self.prop_variable),
00874             str(self.parsimony_informative),
00875             str(self.prop_parsimony)
00876         ]
00877         return summary
00878
```

Referenced by amas.AMAS.AminoAcidAlignment.get_summary(), and amas.AMAS.DNAAlignment.get_summary().

Here is the caller graph for this function:


amas.AMAS.AminoAcidAlignment.get
_summary
amas.AMAS.DNAAlignment.get
_summary
amas.AMAS.Alignment.summarize
_alignment

7.1.3.32 summarize_alignment_by_taxa()

```
amas.AMAS.Alignment.summarize_alignment_by_taxa (
                self )
```

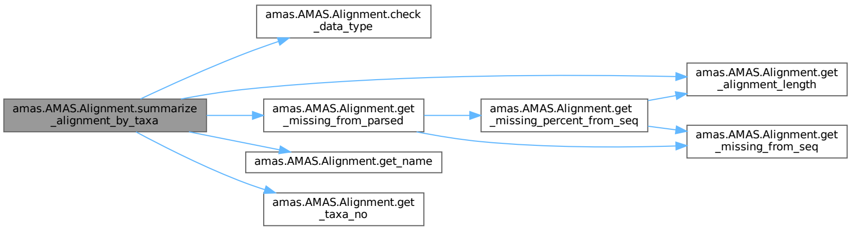
Definition at line 879 of file AMAS.py.

```
00879     def summarize_alignment_by_taxa(self):
00880         # get summary for all taxa/sequences in alignment
00881         per_taxon_summary = []
00882         taxa_no = self.get_taxa_no()
00883         self.missing_records = self.get_missing_from_parsed()
00884         self.length = self.get_alignment_length()
00885         lengths = (self.length for i in range(taxa_no))
00886         name = self.get_name()
00887         names = (name for i in range(taxa_no))
00888         taxa_names = (
00889             taxon.replace(" ", "_").replace(".", "_").replace("'", "")
00890             for taxon, missing_count, missing_percent in self.missing_records
00891         )
00892         missing = (missing_count for taxon, missing_count, missing_percent in self.missing_records)
00893         missing_percent = (missing_percent for taxon, missing_count, missing_percent in
      self.missing_records)
00894         self.check_data_type()
00895         per_taxon_summary = (names, taxa_names, lengths, missing, missing_percent)
00896         zipped = list(zip(*per_taxon_summary))
00897         return zipped
00898
```

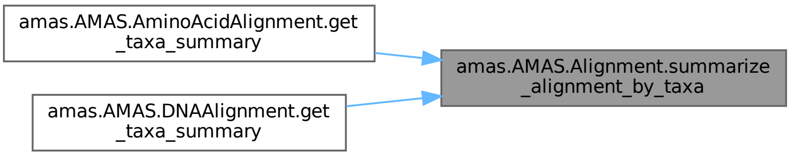
References amas.AMAS.Alignment.check_data_type(), amas.AMAS.Alignment.get_alignment_length(), amas.AMAS.Alignment.get_r amas.AMAS.Alignment.get_name(), amas.AMAS.Alignment.get_taxa_no(), amas.AMAS.Alignment.length, and amas.AMAS.Alignment.missing_records.

Referenced by amas.AMAS.AminoAcidAlignment.get_taxa_summary(), and amas.AMAS.DNAAlignment.get_taxa_summary().

Here is the call graph for this function:


amas.AMAS.Alignment.check _data_type
amas.AMAS.Alignment.get _alignment_length
amas.AMAS.Alignment.summarize _alignment_by_taxa
amas.AMAS.Alignment.get _missing_from_parsed
amas.AMAS.Alignment.get _missing_percent_from_seq
amas.AMAS.Alignment.get _missing_from_seq
amas.AMAS.Alignment.get_name
amas.AMAS.Alignment.get _taxa_no

Here is the caller graph for this function:


amas.AMAS.AminoAcidAlignment.get _taxa_summary
amas.AMAS.DNAAlignment.get _taxa_summary
amas.AMAS.Alignment.summarize _alignment_by_taxa

7.1.4 Member Data Documentation

7.1.4.1 all_matrix_cells

`amas.AMAS.Alignment.all_matrix_cells [static]`

Definition at line 1018 of file AMAS.py.

Referenced by amas.AMAS.Alignment.get_missing_percent().

7.1.4.2 check

`amas.AMAS.Alignment.check`

Definition at line 1077 of file AMAS.py.

7.1.4.3 data_type

`amas.AMAS.Alignment.data_type`

Definition at line 818 of file AMAS.py.

Referenced by amas.AMAS.MetaAlignment.get_alignment_object(), amas.AMAS.MetaAlignment.get_summaries(), amas.AMAS.MetaAlignment.get_taxon_summaries(), amas.AMAS.MetaAlignment.print_nexus(), amas.AMAS.MetaAlignment.print_r and amas.AMAS.MetaAlignment.write_out().

7.1.4.4 in_file

`amas.AMAS.Alignment.in_file`

Definition at line 816 of file AMAS.py.

Referenced by amas.AMAS.FileHandler.__exit__(), amas.AMAS.Alignment.get_aln_input(), and amas.AMAS.Alignment.get_name().

7.1.4.5 in_format

`amas.AMAS.Alignment.in_format`

Definition at line 817 of file AMAS.py.

Referenced by amas.AMAS.MetaAlignment.get_alignment_object(), amas.AMAS.Alignment.get_parsed_aln(), and amas.AMAS.MetaAlignment.write_out().

7.1.4.6 length

`amas.AMAS.Alignment.length`

Definition at line 851 of file AMAS.py.

Referenced by amas.AMAS.Alignment.get_prop_parsimony(), amas.AMAS.Alignment.get_prop_variable(), and amas.AMAS.Alignment.summarize_alignment_by_taxa().

7.1.4.7 matrix

`amas.AMAS.Alignment.matrix`

Definition at line 852 of file AMAS.py.

Referenced by amas.AMAS.Alignment.get_column(), and amas.AMAS.Alignment.get_trim_selection().

7.1.4.8 missing

`amas.AMAS.Alignment.missing`

Definition at line 1023 of file AMAS.py.

Referenced by amas.AMAS.Alignment.get_missing_percent().

7.1.4.9 missing_records

`amas.AMAS.Alignment.missing_records`

Definition at line 858 of file AMAS.py.

Referenced by amas.AMAS.Alignment.get_missing_from_parsed(), and amas.AMAS.Alignment.summarize_alignment_by_taxa().

7.1.4.10 no_missing_ambiguous

`amas.AMAS.Alignment.no_missing_ambiguous`

Definition at line 853 of file AMAS.py.

Referenced by amas.AMAS.Alignment.get_parsimony_informative(), and amas.AMAS.Alignment.get_variable().

7.1.4.11 parsed_aln

`amas.AMAS.Alignment.parsed_aln`

Definition at line 820 of file AMAS.py.

Referenced by amas.AMAS.Alignment.check_data_type(), amas.AMAS.Alignment.get_alignment_length(), amas.AMAS.DNAAlignment.get_atgc_from_parsed(), amas.AMAS.Alignment.get_counts_from_parsed(), amas.AMAS.Alignment.get amas.AMAS.Alignment.get_taxa_no(), and amas.AMAS.Alignment.matrix_creator().

7.1.4.12 parsimony_informative

`amas.AMAS.Alignment.parsimony_informative`

Definition at line 856 of file AMAS.py.

Referenced by amas.AMAS.Alignment.get_prop_parsimony().

7.1.4.13 prop_parsimony

`amas.AMAS.Alignment.prop_parsimony`

Definition at line 857 of file AMAS.py.

7.1.4.14 prop_variable

`amas.AMAS.Alignment.prop_variable`

Definition at line 855 of file AMAS.py.

7.1.4.15 variable_sites

`amas.AMAS.Alignment.variable_sites`

Definition at line 854 of file AMAS.py.

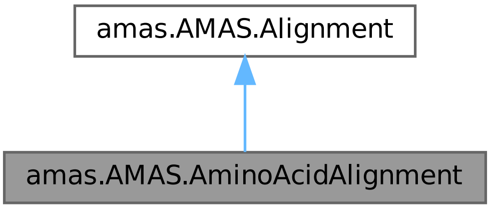
Referenced by amas.AMAS.Alignment.get_prop_variable().

The documentation for this class was generated from the following file:

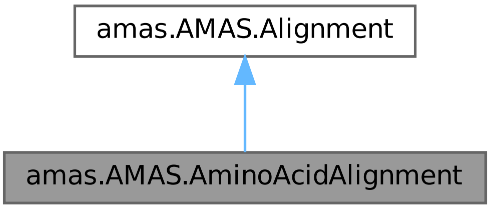
- amas/AMAS.py

7.2 amas.AMAS.AminoAcidAlignment Class Reference

Inheritance diagram for amas.AMAS.AminoAcidAlignment:


amas.AMAS.Alignment
amas.AMAS.AminoAcidAlignment

Collaboration diagram for amas.AMAS.AminoAcidAlignment:


amas.AMAS.Alignment
amas.AMAS.AminoAcidAlignment

Public Member Functions

- get_summary (self)
- get_taxa_summary (self)

Static Public Attributes

- list alphabet = ["A", "C", "D", "E", "F", "G", "H", "I", "K", "L", "M", "N", "P", "Q", "R", "S", "T", "V", "W", "Y", "B", "J", "Z", "X", ".", "*", "-", "?"]
- list missing_ambiguous_chars = ["B", "J", "Z", "X", ".", "*", "-", "?"]
- list missing_chars = ["X", ".", "*", "-", "?"]
- list non_alphabet = ["O"]

7.2.1 Detailed Description

```
Alphabets specific to amino acid alignments
```

Definition at line 1085 of file AMAS.py.

7.2.2 Member Function Documentation

7.2.2.1 get_summary()

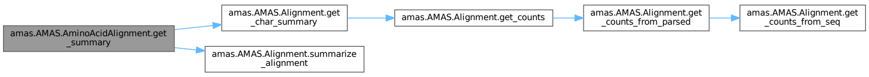
```
amas.AMAS.AminoAcidAlignment.get_summary (
              self )
```

Definition at line 1093 of file AMAS.py.

```
01093     def get_summary(self):
01094         # get alignment summary specific to amino acids
01095         data = self.summarize_alignment()
01096         new_data = data + list(self.get_char_summary()[1])
01097         return new_data
01098
```

References amas.AMAS.Alignment.get_char_summary(), and amas.AMAS.Alignment.summarize_alignment().

Here is the call graph for this function:


amas.AMAS.AminoAcidAlignment.get_summary
amas.AMAS.Alignment.get_char_summary
amas.AMAS.Alignment.summarize_alignment
amas.AMAS.Alignment.get_counts
amas.AMAS.Alignment.get_counts_from_parsed
amas.AMAS.Alignment.get_counts_from_seq

7.2.2.2 get_taxa_summary()

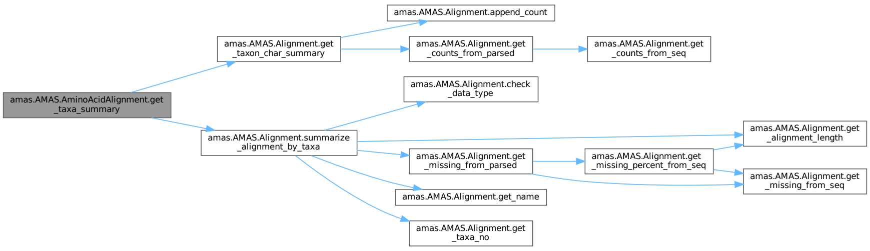
```
amas.AMAS.AminoAcidAlignment.get_taxa_summary (
              self )
```

Definition at line 1099 of file AMAS.py.

```
01099     def get_taxa_summary(self):
01100         # get per-taxon/sequence alignment summary specific to amino acids
01101         data = self.summarize_alignment_by_taxa()
01102         aa_summary = (data, self.get_taxon_char_summary())
01103         zipped_list = list(zip(*aa_summary))
01104         new_data = [list(data_tupl) + chars for data_tupl, chars in zipped_list]
01105         return new_data
01106
```

References amas.AMAS.Alignment.get_taxon_char_summary(), and amas.AMAS.Alignment.summarize_alignment_by_taxa().

Here is the call graph for this function:


amas.AMAS.AminoAcidAlignment.get_taxa_summary
amas.AMAS.Alignment.append_count
amas.AMAS.Alignment.get_taxon_char_summary
amas.AMAS.Alignment.get_counts_from_parsed
amas.AMAS.Alignment.get_counts_from_seq
amas.AMAS.Alignment.check_data_type
amas.AMAS.Alignment.summarize_alignment_by_taxa
amas.AMAS.Alignment.get_alignment_length
amas.AMAS.Alignment.get_missing_from_parsed
amas.AMAS.Alignment.get_missing_percent_from_seq
amas.AMAS.Alignment.get_missing_from_seq
amas.AMAS.Alignment.get_name
amas.AMAS.Alignment.get_taxa_no

7.2.3 Member Data Documentation

7.2.3.1 alphabet

`list amas.AMAS.AminoAcidAlignment.alphabet = ["A", "C", "D", "E", "F", "G", "H", "I", "K", "L", "M", "N", "P", "Q", "R", "S", "T", "V", "W", "Y", "B", "J", "Z", "X", ".", "*", "-", "?"] [static]`

Definition at line 1088 of file AMAS.py.

Referenced by amas.AMAS.Alignment.append_count(), amas.AMAS.Alignment.get_char_summary(), and amas.AMAS.Alignment.get_counts_from_seq().

7.2.3.2 missing_ambiguous_chars

`list amas.AMAS.AminoAcidAlignment.missing_ambiguous_chars = ["B", "J", "Z", "X", ".", "*", "-", "?"] [static]`

Definition at line 1089 of file AMAS.py.

Referenced by amas.AMAS.Alignment.get_site_no_missing_ambiguous().

7.2.3.3 missing_chars

`list amas.AMAS.AminoAcidAlignment.missing_chars = ["X", ".", "*", "-", "?"] [static]`

Definition at line 1090 of file AMAS.py.

Referenced by amas.AMAS.Alignment.get_missing_from_seq(), and amas.AMAS.Alignment.replace_missing().

7.2.3.4 non_alphabet

`list amas.AMAS.AminoAcidAlignment.non_alphabet = ["O"] [static]`

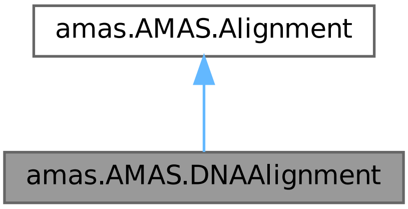
Definition at line 1091 of file AMAS.py.

The documentation for this class was generated from the following file:

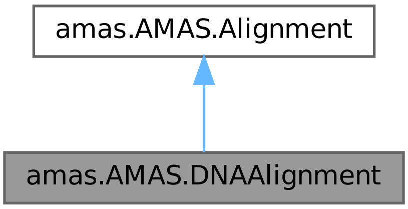
- amas/AMAS.py

7.3 amas.AMAS.DNAAlignment Class Reference

Inheritance diagram for amas.AMAS.DNAAlignment:


amas.AMAS.Alignment
amas.AMAS.DNAAlignment

Collaboration diagram for amas.AMAS.DNAAlignment:


amas.AMAS.Alignment
amas.AMAS.DNAAlignment

Public Member Functions

- get_summary (self)
- get_taxa_summary (self)
- get_atgc_content (self)
- get_list_from_atgc (self)
- get_atgc_from_parsed (self)
- get_atgc_from_seq (self, seq)

Static Public Attributes

- list alphabet = ["A", "C", "G", "T", "K", "M", "R", "Y", "S", "W", "B", "V", "H", "D", "X", "N", "O", "-", "?"]
- list missing_ambiguous_chars = ["K", "M", "R", "Y", "S", "W", "B", "V", "H", "D", "X", "N", "O", "-", "?"]
- list missing_chars = ["X", "N", "O", "-", "?"]
- list non_alphabet = ["E", "F", "I", "L", "P", "Q", "J", "Z", ".", "∗"]

7.3.1 Detailed Description

```
Alphabets specific to DNA alignments
```

Definition at line 1107 of file AMAS.py.

7.3.2 Member Function Documentation

7.3.2.1 get_atgc_content()

```
amas.AMAS.DNAAlignment.get_atgc_content (
                self )
```

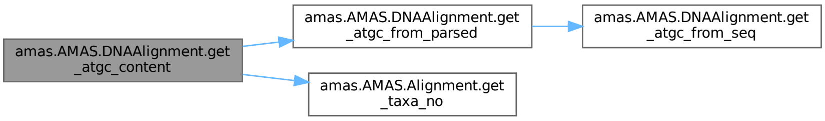
Definition at line 1129 of file AMAS.py.

```
01129     def get_atgc_content(self):
01130         # get AC and GC contents for all sequences
01131         # AT content is the first element of AT, GC content tuple
01132         # returned by get_atgc_from_seq()
01133         atgc_records = self.get_atgc_from_parsed()
01134         at_content = round(sum(atgc[0] for taxon, atgc in atgc_records) / self.get_taxa_no(), 3)
01135         gc_content = round(1 - float(at_content), 3)
01136 
01137         atgc_content = [str(at_content), str(gc_content)]
01138         return atgc_content
01139 
```

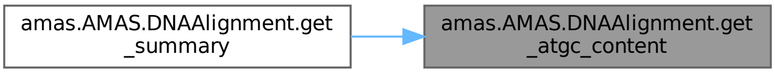
References amas.AMAS.DNAAlignment.get_atgc_from_parsed(), and amas.AMAS.Alignment.get_taxa_no().

Referenced by amas.AMAS.DNAAlignment.get_summary().

Here is the call graph for this function:


amas.AMAS.DNAAlignment.get
_atgc_content
amas.AMAS.DNAAlignment.get
_atgc_from_parsed
amas.AMAS.DNAAlignment.get
_atgc_from_seq
amas.AMAS.Alignment.get
_taxa_no

Here is the caller graph for this function:


amas.AMAS.DNAAlignment.get
_summary
amas.AMAS.DNAAlignment.get
_atgc_content

7.3.2.2 get_atgc_from_parsed()

```
amas.AMAS.DNAAlignment.get_atgc_from_parsed (
                self )
```

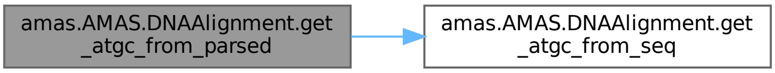
Definition at line 1144 of file AMAS.py.

```
01144     def get_atgc_from_parsed(self):
01145         # get AT and GC contents from parsed alignment dictionary
01146         # return a list of tuples with taxon name, AT content, and GC content
01147         return sorted([(taxon, self.get_atgc_from_seq(seq)) for taxon, seq in
      self.parsed_aln.items()])
01148 
```

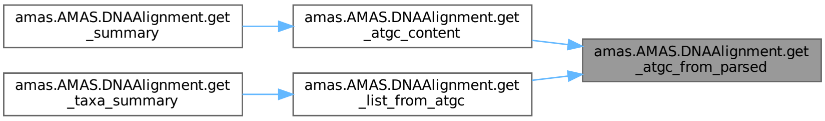
References amas.AMAS.DNAAlignment.get_atgc_from_seq(), and amas.AMAS.Alignment.parsed_aln.

Referenced by amas.AMAS.DNAAlignment.get_atgc_content(), and amas.AMAS.DNAAlignment.get_list_from_atgc().

Here is the call graph for this function:


amas.AMAS.DNAAlignment.get_atgc_from_parsed
amas.AMAS.DNAAlignment.get_atgc_from_seq

Here is the caller graph for this function:


amas.AMAS.DNAAlignment.get_summary
amas.AMAS.DNAAlignment.get_taxa_summary
amas.AMAS.DNAAlignment.get_atgc_content
amas.AMAS.DNAAlignment.get_list_from_atgc
amas.AMAS.DNAAlignment.get_atgc_from_parsed

7.3.2.3 get_atgc_from_seq()

```
amas.AMAS.DNAAlignment.get_atgc_from_seq (
                self,
                seq )
```

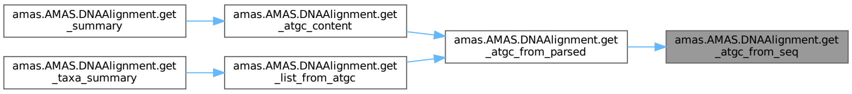
Definition at line 1149 of file AMAS.py.

```
01149     def get_atgc_from_seq(self, seq):
01150         # get AT and GC contents from individual sequences
01151 
01152         at_count = seq.count("A") + seq.count("T") + seq.count("W")
01153         gc_count = seq.count("G") + seq.count("C") + seq.count("S")
01154 
01155         try:
01156             at_content = round(at_count / (at_count + gc_count), 3)
01157             gc_content = round(1 - float(at_content), 3)
01158 
01159         except ZeroDivisionError:
01160             at_content = 0
01161             gc_content = 0
01162 
01163         return at_content, gc_content
```

```
01164
```

Referenced by amas.AMAS.DNAAlignment.get_atgc_from_parsed().

Here is the caller graph for this function:


amas.AMAS.DNAAlignment.get
_summary
amas.AMAS.DNAAlignment.get
_atgc_content
amas.AMAS.DNAAlignment.get
_taxa_summary
amas.AMAS.DNAAlignment.get
_list_from_atgc
amas.AMAS.DNAAlignment.get
_atgc_from_parsed
amas.AMAS.DNAAlignment.get
_atgc_from_seq

7.3.2.4 get_list_from_atgc()

```
amas.AMAS.DNAAlignment.get_list_from_atgc (
                self )
```

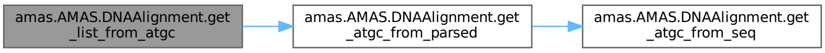
Definition at line 1140 of file AMAS.py.

```
01140     def get_list_from_atgc(self):
01141         records = (atgc for taxon, atgc in self.get_atgc_from_parsed())
01142         return records
01143
```

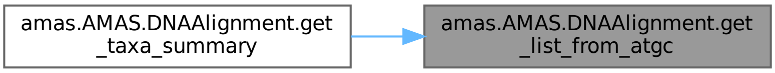
References amas.AMAS.DNAAlignment.get_atgc_from_parsed().

Referenced by amas.AMAS.DNAAlignment.get_taxa_summary().

Here is the call graph for this function:


amas.AMAS.DNAAlignment.get
_list_from_atgc
amas.AMAS.DNAAlignment.get
_atgc_from_parsed
amas.AMAS.DNAAlignment.get
_atgc_from_seq

Here is the caller graph for this function:


amas.AMAS.DNAAlignment.get
_taxa_summary
amas.AMAS.DNAAlignment.get
_list_from_atgc

7.3.2.5 get_summary()

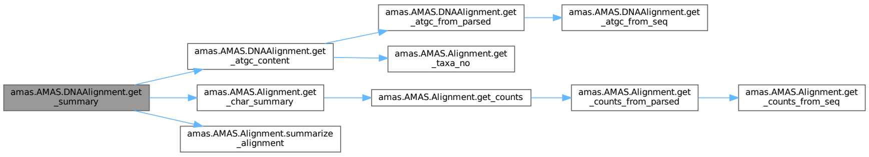
```
amas.AMAS.DNAAlignment.get_summary (
                self )
```

Definition at line 1115 of file AMAS.py.

```
01115     def get_summary(self):
01116         # get alignment summarry specific to nucleotide
01117         data = self.summarize_alignment()
01118         new_data = data + self.get_atgc_content() + list(self.get_char_summary()[1])
01119         return new_data
01120
```

References amas.AMAS.DNAAlignment.get_atgc_content(), amas.AMAS.Alignment.get_char_summary(), and amas.AMAS.Alignment.summarize_alignment().

Here is the call graph for this function:


amas.AMAS.DNAAlignment.get
_summary
amas.AMAS.DNAAlignment.get
_atgc_content
amas.AMAS.Alignment.get
_char_summary
amas.AMAS.Alignment.summarize
_alignment
amas.AMAS.DNAAlignment.get
_atgc_from_parsed
amas.AMAS.Alignment.get
_taxa_no
amas.AMAS.Alignment.get_counts
amas.AMAS.DNAAlignment.get
_atgc_from_seq
amas.AMAS.Alignment.get
_counts_from_parsed
amas.AMAS.Alignment.get
_counts_from_seq

7.3.2.6 get_taxa_summary()

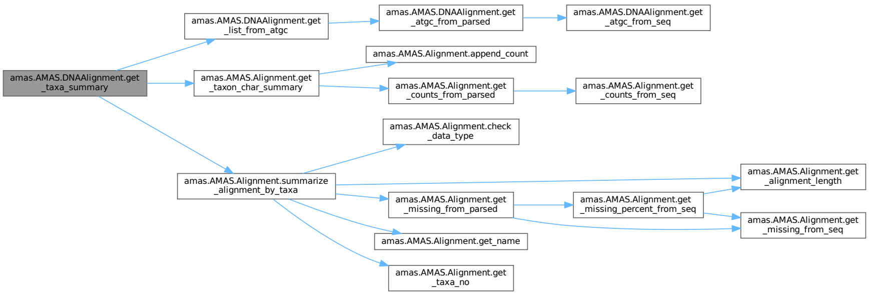
```
amas.AMAS.DNAAlignment.get_taxa_summary (
                self )
```

Definition at line 1121 of file AMAS.py.

```
01121     def get_taxa_summary(self):
01122         # get per-taxon/sequence alignment summary specific to nucleotides
01123         data = self.summarize_alignment_by_taxa()
01124         dna_summary = (data, self.get_list_from_atgc(), self.get_taxon_char_summary())
01125         zipped_list = list(zip(*dna_summary))
01126         new_data = [list(data_tupl) + list(atgc) + chars for data_tupl, atgc, chars in zipped_list]
01127         return new_data
01128
```

References amas.AMAS.DNAAlignment.get_list_from_atgc(), amas.AMAS.Alignment.get_taxon_char_summary(), and amas.AMAS.Alignment.summarize_alignment_by_taxa().

Here is the call graph for this function:


amas.AMAS.DNAAlignment.get
_taxa_summary
amas.AMAS.DNAAlignment.get
_list_from_atgc
amas.AMAS.Alignment.get
_taxon_char_summary
amas.AMAS.Alignment.summarize
_alignment_by_taxa
amas.AMAS.DNAAlignment.get
_atgc_from_parsed
amas.AMAS.Alignment.append_count
amas.AMAS.Alignment.get
_counts_from_parsed
amas.AMAS.Alignment.check
_data_type
amas.AMAS.Alignment.get
_missing_from_parsed
amas.AMAS.Alignment.get_name
amas.AMAS.Alignment.get
_taxa_no
amas.AMAS.DNAAlignment.get
_atgc_from_seq
amas.AMAS.Alignment.get
_counts_from_seq
amas.AMAS.Alignment.get
_missing_percent_from_seq
amas.AMAS.Alignment.get
_alignment_length
amas.AMAS.Alignment.get
_missing_from_seq

7.3.3 Member Data Documentation

7.3.3.1 alphabet

```
list amas.AMAS.DNAAlignment.alphabet = ["A", "C", "G", "T", "K", "M", "R", "Y", "S", "W", "B",
"V", "H", "D", "X", "N", "O", "-", "?"]  [static]
```

Definition at line 1110 of file AMAS.py.

Referenced by amas.AMAS.Alignment.append_count(), amas.AMAS.Alignment.get_char_summary(), and amas.AMAS.Alignment.get_counts_from_seq().

7.3.3.2 missing_ambiguous_chars

```
list amas.AMAS.DNAAlignment.missing_ambiguous_chars = ["K", "M", "R", "Y", "S", "W", "B", "V",
"H", "D", "X", "N", "O", "-", "?"]  [static]
```

Definition at line 1111 of file AMAS.py.

Referenced by amas.AMAS.Alignment.get_site_no_missing_ambiguous().

7.3.3.3 missing_chars

```
list amas.AMAS.DNAAlignment.missing_chars = ["X", "N", "O", "-", "?"]  [static]
```

Definition at line 1112 of file AMAS.py.

Referenced by amas.AMAS.Alignment.get_missing_from_seq(), and amas.AMAS.Alignment.replace_missing().

7.3.3.4 non_alphabet

```
list amas.AMAS.DNAAlignment.non_alphabet = ["E", "F", "I", "L", "P", "Q", "J", "Z", ".", "*"]
[static]
```

Definition at line 1113 of file AMAS.py.

The documentation for this class was generated from the following file:

- amas/AMAS.py

7.4 amas.AMAS.FileHandler Class Reference

Public Member Functions

- __init__ (self, file_name)
- __enter__ (self)
- __exit__ (self, ∗args)
- get_file_name (self)

Public Attributes

- file_name
- in_file

7.4.1 Detailed Description

```
Define file handle that closes when out of scope
```

Definition at line 518 of file AMAS.py.

7.4.2 Constructor & Destructor Documentation

7.4.2.1 __init__()

```
amas.AMAS.FileHandler.__init__ (
              self,
              file_name )
```

Definition at line 521 of file AMAS.py.

```
00521     def __init__(self, file_name):
00522         self.file_name = file_name
00523
```

7.4.3 Member Function Documentation

7.4.3.1 __enter__()

```
amas.AMAS.FileHandler.__enter__ (
              self )
```

Definition at line 524 of file AMAS.py.

```
00524     def __enter__(self):
00525         try:
00526             self.in_file = open(self.file_name, "r", encoding="utf-8")
00527         except FileNotFoundError:
00528             print("ERROR: File ’" + self.file_name + "’ not found.")
00529             sys.exit()
00530         return self.in_file
00531
```

7.4.3.2 __exit__()

```
amas.AMAS.FileHandler.__exit__ (
              self,
             * args )
```

Definition at line 532 of file AMAS.py.

```
00532     def __exit__(self, *args):
00533         self.in_file.close()
00534
```

References amas.AMAS.FileHandler.in_file, amas.AMAS.FileParser.in_file, and amas.AMAS.Alignment.in_file.

7.4.3.3 get_file_name()

```
amas.AMAS.FileHandler.get_file_name (
              self )
```

Definition at line 535 of file AMAS.py.

```
00535     def get_file_name(self):
00536         return self.file_name
00537
```

References amas.AMAS.FileHandler.file_name.

7.4.4 Member Data Documentation

7.4.4.1 file_name

```
amas.AMAS.FileHandler.file_name
```

Definition at line 522 of file AMAS.py.

Referenced by amas.AMAS.FileHandler.get_file_name().

7.4.4.2 in_file

```
amas.AMAS.FileHandler.in_file
```

Definition at line 526 of file AMAS.py.

Referenced by amas.AMAS.FileHandler.__exit__(), amas.AMAS.Alignment.get_aln_input(), and amas.AMAS.Alignment.get_name().

The documentation for this class was generated from the following file:

- amas/AMAS.py

7.5 amas.AMAS.FileParser Class Reference

Public Member Functions

- __init__ (self, in_file)
- fasta_parse (self)
- phylip_parse (self)
- phylip_interleaved_parse (self)
- nexus_parse (self)
- nexus_interleaved_parse (self)
- translate_ambiguous (self, seq)
- partitions_parse (self)

Public Attributes

- in_file
- in_file_lines

Static Public Attributes

- tax_chars_matches
- name_matches
- seq_matches
- tax_match = match.group(2)
- chars_match = match.group(3)
- list taxa = [ ]
- list sequences = [ ]
- dict records = {}
- int counter = 0
- name_match = match.group(2).replace("\n", "")
- seq_match = match.group(3).replace("\n", "").upper()
- matches
- str seq_match = "".join(seq_match.split())
- str sequence = ""

7.5.1 Detailed Description

```
Parse file contents and return sequences and sequence names
```

Definition at line 538 of file AMAS.py.

7.5.2 Constructor & Destructor Documentation

7.5.2.1 __init__()

```
amas.AMAS.FileParser.__init__ (
              self,
              in_file )
```

Definition at line 541 of file AMAS.py.

```
00541     def __init__(self, in_file):
00542         self.in_file = in_file
00543         with FileHandler(in_file) as handle:
00544             self.in_file_lines = handle.read().rstrip("\r\n")
00545
```

7.5.3 Member Function Documentation

7.5.3.1 fasta_parse()

```
amas.AMAS.FileParser.fasta_parse (
              self )
```

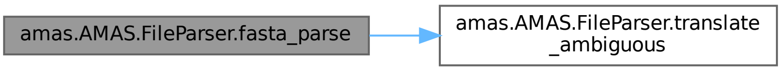
Definition at line 546 of file AMAS.py.

```
00546     def fasta_parse(self):
00547         # use regex to parse names and sequences in sequential fasta files
00548         matches = re.finditer(
00549             r"^>(.*[^$])([^>]*)",
00550             self.in_file_lines, re.MULTILINE
00551         )
00552         records = {}
00553
00554         for match in matches:
```

```
            name_match = match.group(1).replace("\n", "")
            seq_match = match.group(2).replace("\n", "").upper()
            seq_match = self.translate_ambiguous(seq_match)
            records[name_match] = seq_match

        return records

```

References amas.AMAS.FileParser.in_file_lines, and amas.AMAS.FileParser.translate_ambiguous().

Here is the call graph for this function:


amas.AMAS.FileParser.fasta_parse
amas.AMAS.FileParser.translate _ambiguous

7.5.3.2 nexus_interleaved_parse()

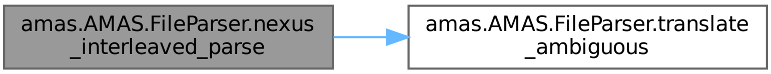
```
amas.AMAS.FileParser.nexus_interleaved_parse (
               self )
```

Definition at line 671 of file AMAS.py.

```
    def nexus_interleaved_parse(self):
        # use regex to parse names and sequences in sequential nexus files
        # find the matrix block
        matches = re.finditer(
            r"(\s+)?(MATRIX\n|matrix\n|MATRIX\r\n|matrix\r\n)(.*?;)",
            self.in_file_lines, re.DOTALL
        )
        # initiate lists for taxa names and sequence strings on separate lines
        taxa = []
        sequences = []
        # initiate a dictionary for the name:sequence records
        records = {}

        for match in matches:
            matrix_match = match.group(3)
            # get names and sequences from the matrix block
            seq_matches = re.finditer(
                r"^(\s+)?[’]?(\S+\s\S+|\S+)[’]?\s+([A-Za-z*?.{}-]+)($|\s+\[[0-9]+\]$)",
                matrix_match, re.MULTILINE
            )

            for match in seq_matches:
                name_match = match.group(2)
                if name_match not in taxa:
                    taxa.append(name_match)
                seq_match = match.group(3)

                sequences.append(seq_match)

        # initiate a counter to keep track of sequences strung together
        # from separate lines
        counter = 0

        for taxon_no in range(len(taxa)):

            full_length_sequence = "".join([sequences[index] for index in
range(counter,len(sequences),len(taxa))])
            records[taxa[taxon_no]] = self.translate_ambiguous(full_length_sequence).replace("\n",
"").upper()
            counter += 1

        return records

```

References amas.AMAS.FileParser.in_file_lines, and amas.AMAS.FileParser.translate_ambiguous().

Here is the call graph for this function:


amas.AMAS.FileParser.nexus_interleaved_parse
amas.AMAS.FileParser.translate_ambiguous

7.5.3.3 nexus_parse()

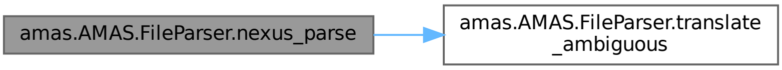
```
amas.AMAS.FileParser.nexus_parse (
                self )
```

Definition at line 645 of file AMAS.py.

```
00645     def nexus_parse(self):
00646         # use regex to parse names and sequences in sequential nexus files
00647         # find the matrix block
00648         matches = re.finditer(
00649             r"(\s+)?(MATRIX\n|matrix\n|MATRIX\r\n|matrix\r\n)(.*?;)",
00650             self.in_file_lines, re.DOTALL
00651         )
00652 
00653         records = {}
00654         # get names and sequences from the matrix block
00655 
00656         for match in matches:
00657             matrix_match = match.group(3)
00658             seq_matches = re.finditer(
00659                 r"^(\s+)?['’]?(\S+\s\S+|\S+)['’]?\s+([A-Za-z*?.{}-]+)($|\s+\[[0-9]+\]$)",
00660                 matrix_match, re.MULTILINE
00661             )
00662 
00663             for match in seq_matches:
00664                 name_match = match.group(2).replace("\n", "")
00665                 seq_match = match.group(3).replace("\n", "").upper()
00666                 seq_match = self.translate_ambiguous(seq_match)
00667                 records[name_match] = seq_match
00668 
00669         return records
00670 
```

References amas.AMAS.FileParser.in_file_lines, and amas.AMAS.FileParser.translate_ambiguous().

Here is the call graph for this function:


amas.AMAS.FileParser.nexus_parse
amas.AMAS.FileParser.translate_ambiguous

7.5.3.4 partitions_parse()

```
amas.AMAS.FileParser.partitions_parse (
                self )
```

Definition at line 731 of file AMAS.py.

```
00731     def partitions_parse(self):
00732         # parse partitions file using regex
00733         # original: `matches = re.finditer(r"^(\s+)?([^ =]+)[ =]+([\\0-9, -]+)", self.in_file_lines,
      re.MULTILINE)`
00734         # new version: more permissive -> handles PartionFinder/RAxML/(IQ-TREE 2)best_scheme.nex
      format partition files
00735         matches = re.finditer(
00736             r"""^[ \t]*                                 # start of line w/ zero-or-more (just)
      whitespaces/tabs
00737                 (
00738                  (?P<nexus>charset[ ]+)                 # case 1: (IQ-TREE 2)best_scheme.nex partition
      directive; partition name
00739                  |
00740                  (?P<raxml>[A-Za-z0-9_+\.]+,[ \t]+)     # case 2: RAxML/RAxML-NG model(+other pars);
      partition name
00741                 )?
00742                 (?P<partition_name>[A-Za-z0-9_@*&()\[\]{}'"%#|-]+) # partition name (handles various
      metcharacters, excludes [\s],<>?/:;~`\^$!)
00743                 [ ]*=[ ]*                               # whitespace-padded (or unpadded) '=':
      (IQ-TREE 2)best_scheme.nex compatabiliy
00744                 (?P<numbers>[\\0-9, -]+)                # position ranges w/ stride (multiple
      intervals; from original regex)
00745                 (?P<nexus_term>[ ]*[;])?                # whitespace-prepended (or unprepended) ';'
      (nexus terminator)
00746             """,
00747             self.in_file_lines,
00748             re.MULTILINE | re.VERBOSE
00749         )
00750
00751         # initiate list to store dictionaries with lists
00752         # of slice positions as values
00753         partitions = []
00754         add_to_partitions = partitions.append
00755
00756         for match in matches:
00757             # initiate dictionary of partition name as key
00758             dict_of_dicts = {}
00759             # and list of dictionaries with slice positions
00760             list_of_dicts = []
00761             add_to_list_of_dicts = list_of_dicts.append
00762             # get parition name and numbers from parsed partition strings
00763             partition_name = match.group('partition_name')
00764             numbers = match.group('numbers')
00765             # remove any whitespace padding '-' (to be consistent with partition-writing format)
00766             numbers = re.sub(r"[ ]*-[ ]*", "-", numbers)
00767             # find all numbers that will be used to parse positions
00768             positions = re.findall(r"([^ ,]+)", numbers)
00769
00770             for position in positions:
00771                 # create dictionary for slicing input sequence
00772                 # conditioning on whether positions are represented
00773                 # by range, range with stride, or single number
00774                 pos_dict = {}
00775
00776                 if "-" in position:
00777                     m = re.search(r"([0-9]+)-([0-9]+)", position)
00778                     pos_dict["start"] = int(m.group(1)) - 1
00779                     pos_dict["stop"] = int(m.group(2))
00780                 else:
00781                     pos_dict["start"] = int(position) - 1
00782                     pos_dict["stop"] = int(position)
00783
00784                 if "\\" in position:
00785                     # Note: the value of `N` in `...\N` isn't read: the script simply assumes `N` is
      consistent with the number of
00786                     # increments per interval when the alignment is parsed with a stride of 3
      (designating each cpos).
00787                     # E.g. For the partition file:
00788                     #       ...`1-N\2`
00789                     #       ...`2-N\2`
00790                     #       ...`(N+1)-M\2`
00791                     #       ...`(N+2)-M\2`
00792                     # 3'cpos are ignored due to the absence of intervals `3-N...', `(N+3)-M...', not
      because the associated stride values are`\2`
00793                     pos_dict["stride"] = 3
00794                 elif "\\" not in position:
00795                     pos_dict["stride"] = 1
00796
00797                 add_to_list_of_dicts(pos_dict)
00798
00799             dict_of_dicts[partition_name] = list_of_dicts
00800             add_to_partitions(dict_of_dicts)
00801
00802         return partitions
00803
00804
```

References amas.AMAS.FileParser.in_file_lines.

7.5.3.5 phylip_interleaved_parse()

```
amas.AMAS.FileParser.phylip_interleaved_parse (
                self )
```

Definition at line 579 of file AMAS.py.

```
00579     def phylip_interleaved_parse(self):
```

7.5.3.6 phylip_parse()

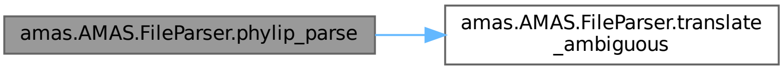
```
amas.AMAS.FileParser.phylip_parse (
                self )
```

Definition at line 562 of file AMAS.py.

```
00562     def phylip_parse(self):
00563         # use regex to parse names and sequences in sequential phylip files
00564         matches = re.finditer(
00565             r"^(\s+)?(\S+)\s+([A-Za-z*?.{}-]+)",
00566             self.in_file_lines, re.MULTILINE
00567         )
00568 
00569         records = {}
00570 
00571         for match in matches:
00572             name_match = match.group(2).replace("\n", "")
00573             seq_match = match.group(3).replace("\n", "").upper()
00574             seq_match = self.translate_ambiguous(seq_match)
00575             records[name_match] = seq_match
00576 
00577         return records
00578 
```

References amas.AMAS.FileParser.in_file_lines, and amas.AMAS.FileParser.translate_ambiguous().

Here is the call graph for this function:


amas.AMAS.FileParser.phylip_parse
amas.AMAS.FileParser.translate
_ambiguous

7.5.3.7 translate_ambiguous()

```
amas.AMAS.FileParser.translate_ambiguous (
                self,
                seq )
```

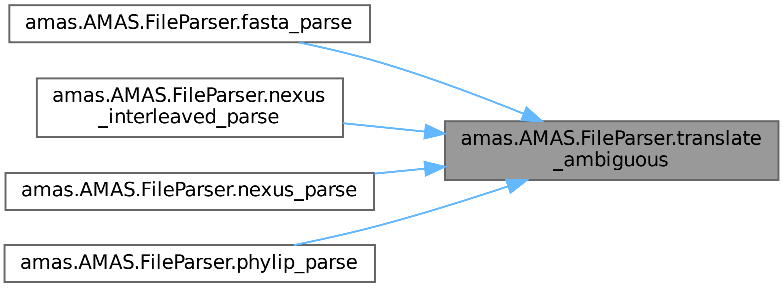
Definition at line 712 of file AMAS.py.

```
00712     def translate_ambiguous(self, seq):
00713         # translate ambiguous characters from curly bracket format
00714         # to single letter format
00715         # also remove spaces from sequences
00716         seq = seq.replace("{GT}", "K")
00717         seq = seq.replace("{AC}", "M")
00718         seq = seq.replace("{AG}", "R")
```

```
        seq = seq.replace("{CT}", "Y")
        seq = seq.replace("{CG}", "S")
        seq = seq.replace("{AT}", "W")
        seq = seq.replace("{CGT}", "B")
        seq = seq.replace("{ACG}", "V")
        seq = seq.replace("{ACT}", "H")
        seq = seq.replace("{AGT}", "D")
        seq = seq.replace("{GATC}", "N")
        seq = seq.replace(" ", "")

        return seq

```

Referenced by amas.AMAS.FileParser.fasta_parse(), amas.AMAS.FileParser.nexus_interleaved_parse(), amas.AMAS.FileParser.nexus_parse(), and amas.AMAS.FileParser.phylip_parse().

Here is the caller graph for this function:


amas.AMAS.FileParser.fasta_parse
amas.AMAS.FileParser.nexus_interleaved_parse
amas.AMAS.FileParser.nexus_parse
amas.AMAS.FileParser.phylip_parse
amas.AMAS.FileParser.translate_ambiguous

7.5.4 Member Data Documentation

7.5.4.1 chars_match

`amas.AMAS.FileParser.chars_match = match.group(3)  [static]`

Definition at line 596 of file AMAS.py.

7.5.4.2 counter

`int amas.AMAS.FileParser.counter = 0  [static]`

Definition at line 605 of file AMAS.py.

7.5.4.3 in_file

`amas.AMAS.FileParser.in_file`

Definition at line 542 of file AMAS.py.

Referenced by amas.AMAS.FileHandler.__exit__(), amas.AMAS.Alignment.get_aln_input(), and amas.AMAS.Alignment.get_name().

7.5.4.4 in_file_lines

`amas.AMAS.FileParser.in_file_lines`

Definition at line 544 of file AMAS.py.

Referenced by amas.AMAS.FileParser.fasta_parse(), amas.AMAS.FileParser.nexus_interleaved_parse(), amas.AMAS.FileParser.nexus_parse(), amas.AMAS.FileParser.partitions_parse(), and amas.AMAS.FileParser.phylip_parse().

7.5.4.5 matches

`amas.AMAS.FileParser.matches  [static]`

Initial value:

```
=  re.finditer(
                r"(^(\s+)?(\S+)( ){2,}|^\s+)([ A-Za-z*?.{}-]+)",
                self.in_file_lines, re.MULTILINE
            )
```

Definition at line 619 of file AMAS.py.

7.5.4.6 name_match

`amas.AMAS.FileParser.name_match = match.group(2).replace("\n", "")  [static]`

Definition at line 608 of file AMAS.py.

7.5.4.7 name_matches

`amas.AMAS.FileParser.name_matches  [static]`

Initial value:

```
=  re.finditer(
            r"^(\s+)?(\S+)[ \t]+[A-Za-z*?.{}-]+",
            self.in_file_lines, re.MULTILINE
        )
```

Definition at line 585 of file AMAS.py.

7.5.4.8 records

`dict amas.AMAS.FileParser.records = {}  [static]`

Definition at line 602 of file AMAS.py.

7.5.4.9 seq_match [1/2]

`amas.AMAS.FileParser.seq_match = match.group(3).replace("\n", "").upper()  [static]`

Definition at line 612 of file AMAS.py.

7.5.4.10 seq_match [2/2]

```
str amas.AMAS.FileParser.seq_match = "".join(seq_match.split())  [static]
```

Definition at line 631 of file AMAS.py.

7.5.4.11 seq_matches

```
amas.AMAS.FileParser.seq_matches  [static]
```

Initial value:

```
=  re.finditer(
            r"(^(\s+)?\S+[ \t]+|^)([A-Za-z*?.{}-]+)$",
            self.in_file_lines, re.MULTILINE
        )
```

Definition at line 589 of file AMAS.py.

7.5.4.12 sequence

```
str amas.AMAS.FileParser.sequence = ""  [static]
```

Definition at line 636 of file AMAS.py.

7.5.4.13 sequences

```
list amas.AMAS.FileParser.sequences = []  [static]
```

Definition at line 600 of file AMAS.py.

7.5.4.14 tax_chars_matches

```
amas.AMAS.FileParser.tax_chars_matches  [static]
```

Initial value:

```
=  re.finditer(
            r"^(\s+)?([0-9]+)[ \t]+([0-9]+)",
            self.in_file_lines, re.MULTILINE
        )
```

Definition at line 581 of file AMAS.py.

7.5.4.15 tax_match

```
amas.AMAS.FileParser.tax_match = match.group(2)  [static]
```

Definition at line 595 of file AMAS.py.

7.5.4.16 taxa

```
list amas.AMAS.FileParser.taxa = [ ]  [static]
```

Definition at line 599 of file AMAS.py.

The documentation for this class was generated from the following file:

- amas/AMAS.py

7.6 amas.AMAS.MetaAlignment Class Reference

Public Member Functions

- __init__ (self, **kwargs)
- translate_dna_to_aa (self, seq, translation_table, frame)
- translate_dict (self, source_dict)
- get_translated (self, translation_table, reading_frame)
- trim_dict (self, alignment)
- get_trimmed (self, trim_fraction, parsimony_check)
- remove_unknown_chars (self, seq)
- remove_empty_sequences (self, split_alignment)
- get_partitions (self, partitions_file)
- get_alignment_object (self, alignment)
- get_alignment_objects (self)
- get_parsed_alignments (self)
- get_partitioned (self, partitions_file)
- get_summaries (self)
- summarize_alignments (self, alignment)
- get_taxon_summaries (self)
- summarize_alignments_taxa (self, alignment)
- write_summaries (self, file_name)
- write_taxa_summaries (self)
- get_replicate (self, no_replicates, no_loci)
- get_concatenated (self, alignments)
- remove_from_alignment (self, alignment, species_to_remove_set, index)
- remove_taxa (self, species_to_remove_set)
- print_fasta (self, source_dict)
- print_phylip (self, source_dict)
- print_phylip_int (self, source_dict)
- print_nexus (self, source_dict)
- print_nexus_int (self, source_dict)
- natural_sort (self, a_list)
- print_unspecified_partitions (self, data_type, codons)
- print_nexus_partitions (self, data_type, codons)
- print_iqtree_nexus_partitions (self, data_type, codons)
- print_raxml_partitions (self, data_type, codons)
- replace_string_in_file (self, file_name, old_string, new_string)
- write_partitions (self, file_name, part_format, data_type, codons)
- get_extension (self, file_format)
- get_metapartition_extension (self, file_format)
- file_overwrite_error (self, file_name)

- write_formatted_file (self, file_format, file_name, alignment)
- get_alignment_name (self, i, extension)
- get_alignment_name_no_ext (self, i)
- write_concat (self, file_format)
- write_convert (self, index, alignment, file_format, extension)
- write_replicate (self, index, alignment, file_format, extension)
- write_split (self, item, file_format, extension)
- write_reduced (self, file_format, extension)
- write_translated (self, index, alignment, file_format, extension)
- write_trimmed (self, index, alignment, file_format, extension)
- write_metapartitions (self, file_format)
- write_out (self, action, file_format)

Public Attributes

- in_files
- in_format
- data_type
- command
- concat_out
- using_metapartitions
- check_align
- cores
- by_taxon_summary
- no_sup_aln_name
- no_mpan
- codons
- no_replicates
- no_loci
- split
- remove_empty
- prepend_label
- species_to_remove
- species_to_remove_set
- reduced_file_prefix
- check_taxa
- reading_frame
- genetic_code
- trim_fraction
- trim_out
- parsimony_check
- alignment_objects
- parsed_alignments
- codes_list
- gencode_NCBI_1
- gencode_NCBI_2
- gencode_NCBI_3
- gencode_NCBI_4
- gencode_NCBI_5
- gencode_NCBI_6
- gencode_NCBI_9
- gencode_NCBI_10
- gencode_NCBI_11
- gencode_NCBI_12

- gencode_NCBI_13
- gencode_NCBI_14
- gencode_NCBI_16
- gencode_NCBI_21
- gencode_NCBI_22
- gencode_NCBI_23
- gencode_NCBI_24
- gencode_NCBI_25
- gencode_NCBI_26
- codes

7.6.1 Detailed Description

```
Class of multiple sequence alignments
```

Definition at line 1165 of file AMAS.py.

7.6.2 Constructor & Destructor Documentation

7.6.2.1 __init__()

```
amas.AMAS.MetaAlignment.__init__ (
                self,
                ** kwargs )
```

Definition at line 1168 of file AMAS.py.

```
01168     def __init__(self, **kwargs):
01169         # set defaults and get values from kwargs
01170         self.in_files = kwargs.get("in_files")
01171         self.in_format = kwargs.get("in_format")
01172         self.data_type = kwargs.get("data_type")
01173         self.command = kwargs.get("command")
01174         self.concat_out = kwargs.get("concat_out", "concatenated.out")
01175         self.using_metapartitions = False
01176         self.check_align = kwargs.get("check_align", False)
01177         self.cores = kwargs.get("cores")
01178         self.by_taxon_summary = kwargs.get("by_taxon_summary")
01179         self.no_sup_aln_name = False
01180         self.no_mpan = False
01181 
01182         if self.command == "concat":
01183             self.codons = kwargs.get("codons", "none")
01184             if self.data_type == "aa" and self.codons != "none":
01185                 print("ERROR: when option -d|--data-type is set to ’aa’, option -n|--codons must be
      set to ’none’.")
01186                 sys.exit(1)
01187 
01188         if self.command == "replicate":
01189             self.no_replicates = kwargs.get("replicate_args")[0]
01190             self.no_loci = kwargs.get("replicate_args")[1]
01191 
01192         if self.command == "split":
01193             self.split = kwargs.get("split_by")
01194             self.remove_empty = kwargs.get("remove_empty", False)
01195             self.no_sup_aln_name = kwargs.get("no_sup_aln_name", False)
01196 
01197         if self.command == "metapartitions":
01198             self.using_metapartitions = True
01199             self.split = kwargs.get("split_by")
01200             self.remove_empty = kwargs.get("remove_empty", False)
01201             self.no_sup_aln_name = kwargs.get("no_sup_aln_name", False)
01202             self.no_mpan = kwargs.get("no_mpan", False)
01203             self.prepend_label = kwargs.get("prepend_label")
01204             if self.prepend_label is not None and isinstance(self.prepend_label, str):
01205                 self.prepend_label = self.prepend_label + "_"
01206             else:
```

```
                self.prepend_label = ""

        if self.command == "remove":
            self.species_to_remove = kwargs.get("taxa_to_remove")
            self.species_to_remove_set = set(self.species_to_remove)
            self.reduced_file_prefix = kwargs.get("out_prefix")
            self.check_taxa = kwargs.get("check_taxa", False)

        if self.command == "translate":
            self.reading_frame = kwargs.get("reading_frame")
            self.genetic_code = kwargs.get("genetic_code")

        if self.command == "trim":
            self.trim_fraction = kwargs.get("trim_fraction")
            self.trim_out = kwargs.get("trim_out")
            self.parsimony_check = kwargs.get("parsimony_check", False)

        self.alignment_objects = self.get_alignment_objects()
        self.parsed_alignments = self.get_parsed_alignments()

        # The code list:
        self.codes_list = """
         1. The Standard Code
         2. The Vertebrate Mitochondrial Code
         3. The Yeast Mitochondrial Code
         4. The Mold, Protozoan, and Coelenterate Mitochondrial Code and the Mycoplasma/Spiroplasma Code
         5. The Invertebrate Mitochondrial Code
         6. The Ciliate, Dasycladacean and Hexamita Nuclear Code
         9. The Echinoderm and Flatworm Mitochondrial Code
        10. The Euplotid Nuclear Code
        11. The Bacterial, Archaeal and Plant Plastid Code
        12. The Alternative Yeast Nuclear Code
        13. The Ascidian Mitochondrial Code
        14. The Alternative Flatworm Mitochondrial Code
        16. Chlorophycean Mitochondrial Code
        21. Trematode Mitochondrial Code
        22. Scenedesmus obliquus Mitochondrial Code
        23. Thraustochytrium Mitochondrial Code
        24. Pterobranchia Mitochondrial Code
        25. Candidate Division SR1 and Gracilibacteria Code
        26. Pachysolen tannophilus Nuclear Code
        """

        # 1: The Standard Code
        self.gencode_NCBI_1 = {
        "TTT" : "F", # Phe
        "TCT" : "S", # Ser
        "TAT" : "Y", # Tyr
        "TGT" : "C", # Cys
        "TTC" : "F", # Phe
        "TCC" : "S", # Ser
        "TAC" : "Y", # Tyr
        "TGC" : "C", # Cys
        "TTA" : "L", # Leu
        "TCA" : "S", # Ser
        "TAA" : "*", # Ter
        "TGA" : "*", # Ter
        "TTG" : "L", # Leu i
        "TCG" : "S", # Ser
        "TAG" : "*", # Ter
        "TGG" : "W", # Trp
        "CTT" : "L", # Leu
        "CCT" : "P", # Pro
        "CAT" : "H", # His
        "CGT" : "R", # Arg
        "CTC" : "L", # Leu
        "CCC" : "P", # Pro
        "CAC" : "H", # His
        "CGC" : "R", # Arg
        "CTA" : "L", # Leu
        "CCA" : "P", # Pro
        "CAA" : "Q", # Gln
        "CGA" : "R", # Arg
        "CTG" : "L", # Leu i
        "CCG" : "P", # Pro
        "CAG" : "Q", # Gln
        "CGG" : "R", # Arg
        "ATT" : "I", # Ile
        "ACT" : "T", # Thr
        "AAT" : "N", # Asn
        "AGT" : "S", # Ser
        "ATC" : "I", # Ile
        "ACC" : "T", # Thr
        "AAC" : "N", # Asn
        "AGC" : "S", # Ser
        "ATA" : "I", # Ile
        "ACA" : "T", # Thr
        "AAA" : "K", # Lys
        "AGA" : "R", # Arg
        "ATG" : "M", # Met i
        "ACG" : "T", # Thr
        "AAG" : "K", # Lys
        "AGG" : "R", # Arg
        "GTT" : "V", # Val
        "GCT" : "A", # Ala
        "GAT" : "D", # Asp
        "GGT" : "G", # Gly
        "GTC" : "V", # Val
        "GCC" : "A", # Ala
        "GAC" : "D", # Asp
        "GGC" : "G", # Gly
        "GTA" : "V", # Val
        "GCA" : "A", # Ala
        "GAA" : "E", # Glu
        "GGA" : "G", # Gly
        "GTG" : "V", # Val
        "GCG" : "A", # Ala
        "GAG" : "E", # Glu
        "GGG" : "G", # Gly
        "---" : "-", # Gap
        "???" : "?", # Unk
        "NNN" : "X", # Unk
        }

        # 2: The Vertebrate Mitochondrial Code
        self.gencode_NCBI_2 = self.gencode_NCBI_1.copy()
        self.gencode_NCBI_2["AGA"] = "*" # Ter
        self.gencode_NCBI_2["AGG"] = "*" # Ter
        self.gencode_NCBI_2["ATA"] = "M" # Met
        self.gencode_NCBI_2["TGA"] = "W" # Trp

        # 3: The Yeast Mitochondrial Code
        self.gencode_NCBI_3 = self.gencode_NCBI_1.copy()
        self.gencode_NCBI_3["ATA"] = "M" # Met
        self.gencode_NCBI_3["CTT"] = "T" # Thr
        self.gencode_NCBI_3["CTC"] = "T" # Thr
        self.gencode_NCBI_3["CTA"] = "T" # Thr
        self.gencode_NCBI_3["CTG"] = "T" # Thr
        self.gencode_NCBI_3["TGA"] = "W" # Trp

        del self.gencode_NCBI_3["CGA"]
        del self.gencode_NCBI_3["CGC"]

        # 4: The Mold, Protozoan, and Coelenterate Mitochondrial Code and the Mycoplasma/Spiroplasma Code
        self.gencode_NCBI_4 = self.gencode_NCBI_1.copy()
        self.gencode_NCBI_4["TGA"] = "W" # Trp

        # 5: The Invertebrate Mitochondrial Code
        self.gencode_NCBI_5 = self.gencode_NCBI_1.copy()
        self.gencode_NCBI_5["AGA"] = "S" # Ser
        self.gencode_NCBI_5["AGG"] = "S" # Ser
        self.gencode_NCBI_5["ATA"] = "M" # Met
        self.gencode_NCBI_5["TGA"] = "W" # Trp

        # 6: The Ciliate, Dasycladacean and Hexamita Nuclear Code
        self.gencode_NCBI_6 = self.gencode_NCBI_1.copy()
        self.gencode_NCBI_6["TAA"] = "Q" # Gln
        self.gencode_NCBI_6["TAG"] = "Q" # Gln

        # 9: The Echinoderm and Flatworm Mitochondrial Code
        self.gencode_NCBI_9 = self.gencode_NCBI_1.copy()
        self.gencode_NCBI_9["AAA"] = "N" # Asn
        self.gencode_NCBI_9["AGA"] = "S" # Ser
        self.gencode_NCBI_9["AGG"] = "S" # Ser
        self.gencode_NCBI_9["TGA"] = "W" # Trp

        # 10: The Euplotid Nuclear Code
        self.gencode_NCBI_10 = self.gencode_NCBI_1.copy()
        self.gencode_NCBI_10["TGA"] = "C" # Cys

        # 11: The Bacterial, Archaeal and Plant Plastid Code
        self.gencode_NCBI_11 = self.gencode_NCBI_1.copy()

        # 12: The Alternative Yeast Nuclear Code
        self.gencode_NCBI_12 = self.gencode_NCBI_1.copy()
        self.gencode_NCBI_12["CTG"] = "S" # Ser

        # 13: The Ascidian Mitochondrial Code
        self.gencode_NCBI_13 = self.gencode_NCBI_1.copy()
        self.gencode_NCBI_13["AGA"] = "G" # Gly
        self.gencode_NCBI_13["AGG"] = "G" # Gly
        self.gencode_NCBI_13["ATA"] = "M" # Met
        self.gencode_NCBI_13["TGA"] = "W" # Trp

        # 14: The Alternative Flatworm Mitochondrial Code
        self.gencode_NCBI_14 = self.gencode_NCBI_1.copy()
        self.gencode_NCBI_14["AAA"] = "N" # Asn
        self.gencode_NCBI_14["AGA"] = "S" # Ser
        self.gencode_NCBI_14["AGG"] = "S" # Ser
        self.gencode_NCBI_14["TAA"] = "Y" # Tyr
        self.gencode_NCBI_14["TGA"] = "W" # Trp

        # 16: Chlorophycean Mitochondrial Code
        self.gencode_NCBI_16 = self.gencode_NCBI_1.copy()
        self.gencode_NCBI_16["TAG"] = "L" # Leu

        # 21: Trematode Mitochondrial Code
        self.gencode_NCBI_21 = self.gencode_NCBI_1.copy()
        self.gencode_NCBI_21["TGA"] = "W" # Trp
        self.gencode_NCBI_21["ATA"] = "M" # Met
        self.gencode_NCBI_21["AGA"] = "S" # Ser
        self.gencode_NCBI_21["AGG"] = "S" # Ser
        self.gencode_NCBI_21["AAA"] = "N" # Asn

        # 22: Scenedesmus obliquus Mitochondrial Code
        self.gencode_NCBI_22 = self.gencode_NCBI_1.copy()
        self.gencode_NCBI_22["TCA"] = "*" # Ter
        self.gencode_NCBI_22["TAG"] = "L" # Leu

        # 23: Thraustochytrium Mitochondrial Code
        self.gencode_NCBI_23 = self.gencode_NCBI_1.copy()
        self.gencode_NCBI_23["TTA"] = "*" # Ter

        # 24: Pterobranchia Mitochondrial Code
        self.gencode_NCBI_24 = self.gencode_NCBI_1.copy()
        self.gencode_NCBI_24["AGA"] = "S" # Ser
        self.gencode_NCBI_24["AGG"] = "K" # Lys
        self.gencode_NCBI_24["TGA"] = "W" # Trp

        # 25: Candidate Division SR1 and Gracilibacteria Code
        self.gencode_NCBI_25 = self.gencode_NCBI_1.copy()
        self.gencode_NCBI_25["TGA"] = "G" # Gly

        # 26: Pachysolen tannophilus Nuclear Code
        self.gencode_NCBI_26 = self.gencode_NCBI_1.copy()
        self.gencode_NCBI_26["CTG"] = "A" # Ala

        self.codes = {
        1 : self.gencode_NCBI_1,
        2 : self.gencode_NCBI_2,
        3 : self.gencode_NCBI_3,
        4 : self.gencode_NCBI_4,
        5 : self.gencode_NCBI_5,
        6 : self.gencode_NCBI_6,
        9 : self.gencode_NCBI_9,
        10 : self.gencode_NCBI_10,
        11 : self.gencode_NCBI_11,
        12 : self.gencode_NCBI_12,
        13 : self.gencode_NCBI_13,
        14 : self.gencode_NCBI_14,
        16 : self.gencode_NCBI_16,
        21 : self.gencode_NCBI_21,
        22 : self.gencode_NCBI_22,
        23 : self.gencode_NCBI_23,
        24 : self.gencode_NCBI_24,
        25 : self.gencode_NCBI_25,
        26 : self.gencode_NCBI_26
        }

```

7.6.3 Member Function Documentation

7.6.3.1 file_overwrite_error()

amas.AMAS.MetaAlignment.file_overwrite_error (
self,
file_name)

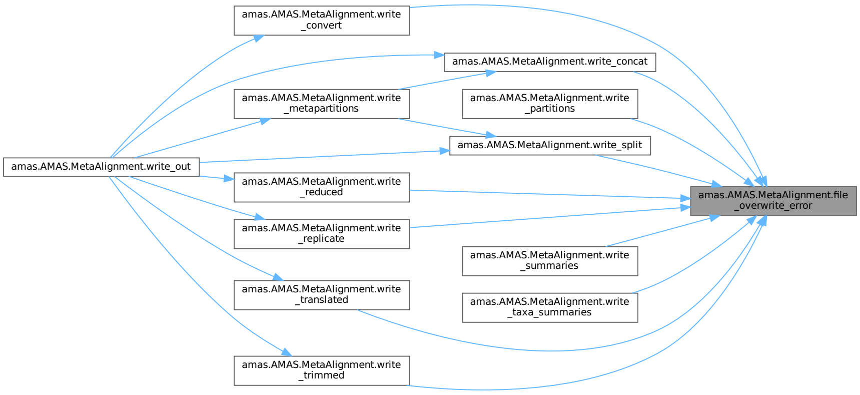
Definition at line 2157 of file AMAS.py.

```
02157     def file_overwrite_error(self, file_name):
```

```
02158         # print warning when overwriting a file
02159         if path.exists(file_name):
02160             print("WARNING: You are overwriting '" + file_name + "'")
02161
```

Referenced by amas.AMAS.MetaAlignment.write_concat(), amas.AMAS.MetaAlignment.write_convert(), amas.AMAS.MetaAlignment
amas.AMAS.MetaAlignment.write_reduced(), amas.AMAS.MetaAlignment.write_replicate(), amas.AMAS.MetaAlignment.write_split()
amas.AMAS.MetaAlignment.write_summaries(), amas.AMAS.MetaAlignment.write_taxa_summaries(), amas.AMAS.MetaAlignment.w
and amas.AMAS.MetaAlignment.write_trimmed().

Here is the caller graph for this function:


amas.AMAS.MetaAlignment.write_out
amas.AMAS.MetaAlignment.write _convert
amas.AMAS.MetaAlignment.write_concat
amas.AMAS.MetaAlignment.write _metapartitions
amas.AMAS.MetaAlignment.write _partitions
amas.AMAS.MetaAlignment.write_split
amas.AMAS.MetaAlignment.write _reduced
amas.AMAS.MetaAlignment.file _overwrite_error
amas.AMAS.MetaAlignment.write _replicate
amas.AMAS.MetaAlignment.write _summaries
amas.AMAS.MetaAlignment.write _translated
amas.AMAS.MetaAlignment.write _taxa_summaries
amas.AMAS.MetaAlignment.write _trimmed

7.6.3.2 get_alignment_name()

amas.AMAS.MetaAlignment.get_alignment_name (
self,
i,
extension)

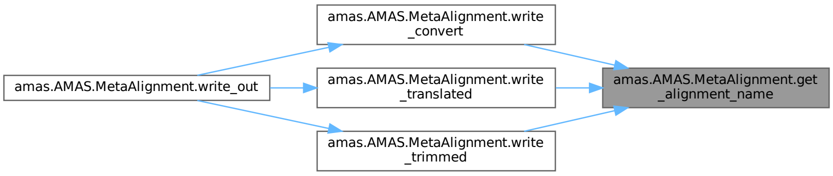
Definition at line 2176 of file AMAS.py.

```
02176     def get_alignment_name(self, i, extension):
02177         # get file name
02178         file_name = self.alignment_objects[i].get_name() + extension
02179
02180         return file_name
02181
```

References amas.AMAS.MetaAlignment.alignment_objects.

Referenced by amas.AMAS.MetaAlignment.write_convert(), amas.AMAS.MetaAlignment.write_translated(), and amas.AMAS.MetaAlignment.write_trimmed().

Here is the caller graph for this function:


amas.AMAS.MetaAlignment.write_out
amas.AMAS.MetaAlignment.write _convert
amas.AMAS.MetaAlignment.write _translated
amas.AMAS.MetaAlignment.write _trimmed
amas.AMAS.MetaAlignment.get _alignment_name

7.6.3.3 get_alignment_name_no_ext()

```
amas.AMAS.MetaAlignment.get_alignment_name_no_ext (
                self,
                i )
```

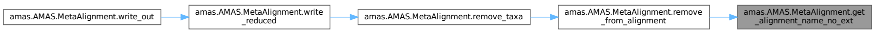
Definition at line 2182 of file AMAS.py.

```
02182     def get_alignment_name_no_ext(self, i):
02183         # get file name without extension
02184         file_name = self.alignment_objects[i].get_name()
02185 
02186         return file_name
02187 
```

References amas.AMAS.MetaAlignment.alignment_objects.

Referenced by amas.AMAS.MetaAlignment.remove_from_alignment().

Here is the caller graph for this function:


amas.AMAS.MetaAlignment.write_out
amas.AMAS.MetaAlignment.write_reduced
amas.AMAS.MetaAlignment.remove_taxa
amas.AMAS.MetaAlignment.remove_from_alignment
amas.AMAS.MetaAlignment.get_alignment_name_no_ext

7.6.3.4 get_alignment_object()

```
amas.AMAS.MetaAlignment.get_alignment_object (
                self,
                alignment )
```

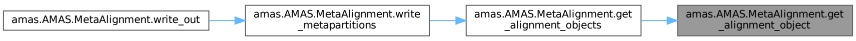
Definition at line 1523 of file AMAS.py.

```
01523     def get_alignment_object(self, alignment):
01524         # parse according to the given alphabet;
01525         # Note:('alignment') <=> `in_file` outside MetaAlignment, e.g.
01526         #
      AminoAcidAlignment(Alignment<-.get_parsed_aln<-.get_aln_input)<-FileParser.__init__(in_file)<-FileHandler(...open(self.
01527         if self.data_type == "aa":
01528             aln = AminoAcidAlignment(alignment, self.in_format, self.data_type)
01529         elif self.data_type == "dna":
01530             aln = DNAAlignment(alignment, self.in_format, self.data_type)
01531         return aln
01532 
```

References amas.AMAS.Alignment.data_type, amas.AMAS.MetaAlignment.data_type, amas.AMAS.Alignment.in_format, and amas.AMAS.MetaAlignment.in_format.

Referenced by amas.AMAS.MetaAlignment.get_alignment_objects().

Here is the caller graph for this function:


amas.AMAS.MetaAlignment.write_out
amas.AMAS.MetaAlignment.write_metapartitions
amas.AMAS.MetaAlignment.get_alignment_objects
amas.AMAS.MetaAlignment.get_alignment_object

7.6.3.5 get_alignment_objects()

```
amas.AMAS.MetaAlignment.get_alignment_objects (
                self )
```

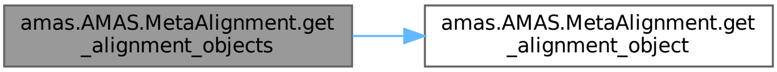
Definition at line 1533 of file AMAS.py.

```
01533     def get_alignment_objects(self):
01534         # get alignment objects on which statistics can be computed
01535         # use multiprocessing if more than one core specified
01536         if int(self.cores) == 1:
01537             alignments = [self.get_alignment_object(alignment) for alignment in self.in_files]
01538         elif int(self.cores) > 1:
01539             pool = mp.Pool(int(self.cores))
01540             alignments = pool.map(self.get_alignment_object, self.in_files)
01541         return alignments
01542 
```

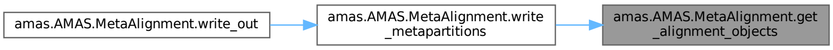
References amas.AMAS.MetaAlignment.cores, amas.AMAS.MetaAlignment.get_alignment_object(), and amas.AMAS.MetaAlignment.in_files.

Referenced by amas.AMAS.MetaAlignment.write_metapartitions().

Here is the call graph for this function:


amas.AMAS.MetaAlignment.get
_alignment_objects
amas.AMAS.MetaAlignment.get
_alignment_object

Here is the caller graph for this function:


amas.AMAS.MetaAlignment.write_out
amas.AMAS.MetaAlignment.write
_metapartitions
amas.AMAS.MetaAlignment.get
_alignment_objects

7.6.3.6 get_concatenated()

```
amas.AMAS.MetaAlignment.get_concatenated (
                self,
                alignments )
```

Definition at line 1746 of file AMAS.py.

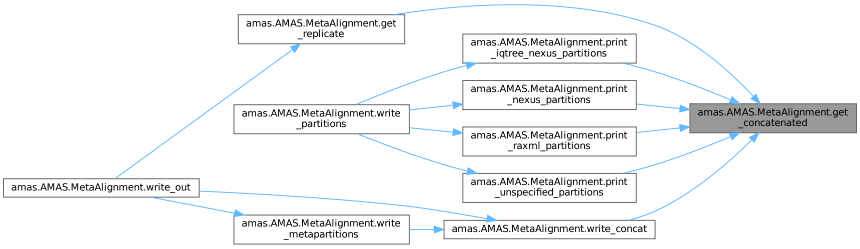
```
01746     def get_concatenated(self, alignments):
01747         # concatenate muntiple input alignments
01748         # create empty dictionary of lists
01749         concatenated = defaultdict(list)
01750 
01751         # first create list of taxa in all alignments
01752         # you need this to insert empty seqs in
01753         # the concatenated alignment
01754         all_taxa = []
01755         for alignment in alignments:
01756             for taxon in alignment.keys():
01757                 if taxon not in all_taxa:
01758                     all_taxa.append(taxon)
01759 
```

```
        # start counters to keep track of partitions
        partition_counter = 1
        position_counter = 1
        # get dict for alignment name and partition
        partitions = {}
        digits_to_pad = len(str(len(alignments)))

        for alignment in alignments:
            # get alignment length from a random taxon
            partition_length = len(alignment[list(alignment.keys())[0]])
            # get base name of each alignment for use when writing partitions file
            # NOTE: the base name here is whatever comes before fist period in the file name
            alignment_name = self.alignment_objects[partition_counter - 1].get_name().split('.')[0]

            if self.using_metapartitions:
                # implementation of option --no-mpan; option --prepend(-label) will assign a string or "" (see class definition)
                if self.no_mpan:
                    # omit original alignment names from the printed partition file
                    partition_name = self.prepend_label + "p" + str(partition_counter).zfill(digits_to_pad)
                else:
                    # keep original alignment names in the printed partition file
                    partition_name = self.prepend_label + "p" + str(partition_counter).zfill(digits_to_pad) + "_" + alignment_name
            else:
                partition_name = "p" + str(partition_counter) + "_" + alignment_name

            start = position_counter
            position_counter += partition_length
            end = position_counter - 1
            partitions[partition_name] = str(start) + "-" + str(end)
            partition_counter += 1

            # get empty sequence if there is missing taxon
            # getting length from first element of list of keys
            # created from the original dict for this alignment
            empty_seq = '?' * partition_length

            for taxon in all_taxa:

                if taxon not in alignment.keys():
                    concatenated[taxon].append(empty_seq)
                else:
                    concatenated[taxon].append(alignment[taxon])

        concatenated = {taxon:”.join(seqs) for taxon, seqs in concatenated.items()}

        return concatenated, partitions

```

References amas.AMAS.MetaAlignment.alignment_objects, amas.AMAS.MetaAlignment.no_mpan, amas.AMAS.MetaAlignment.prep amas.AMAS.MetaAlignment.split, and amas.AMAS.MetaAlignment.using_metapartitions.

Referenced by amas.AMAS.MetaAlignment.get_replicate(), amas.AMAS.MetaAlignment.print_iqtree_nexus_partitions(), amas.AMAS.MetaAlignment.print_nexus_partitions(), amas.AMAS.MetaAlignment.print_raxml_partitions(), amas.AMAS.MetaAlignment.print_unspecified_partitions(), and amas.AMAS.MetaAlignment.write_concat().

Here is the caller graph for this function:


amas.AMAS.MetaAlignment.get_replicate
amas.AMAS.MetaAlignment.print_iqtree_nexus_partitions
amas.AMAS.MetaAlignment.print_nexus_partitions
amas.AMAS.MetaAlignment.write_partitions
amas.AMAS.MetaAlignment.get_concatenated
amas.AMAS.MetaAlignment.print_raxml_partitions
amas.AMAS.MetaAlignment.write_out
amas.AMAS.MetaAlignment.print_unspecified_partitions
amas.AMAS.MetaAlignment.write_metapartitions
amas.AMAS.MetaAlignment.write_concat

7.6.3.7 get_extension()

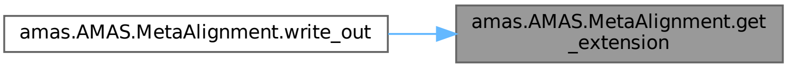
```
amas.AMAS.MetaAlignment.get_extension (
                self,
                file_format )
```

Definition at line 2127 of file AMAS.py.

```
02127     def get_extension(self, file_format):
02128         # get proper extension string
02129         if file_format == "phylip":
02130             extension = "-out.phy"
02131         elif file_format == "phylip-int":
02132             extension = "-out.int-phy"
02133         elif file_format == "fasta":
02134             extension = "-out.fas"
02135         elif file_format == "nexus":
02136             extension = "-out.nex"
02137         elif file_format == "nexus-int":
02138             extension = "-out.int-nex"
02139 
02140         return extension
02141 
```

Referenced by amas.AMAS.MetaAlignment.write_out().

Here is the caller graph for this function:


amas.AMAS.MetaAlignment.write_out
amas.AMAS.MetaAlignment.get_extension

7.6.3.8 get_metapartition_extension()

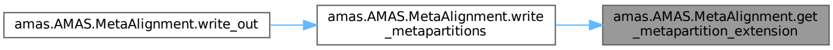
```
amas.AMAS.MetaAlignment.get_metapartition_extension (
                self,
                file_format )
```

Definition at line 2142 of file AMAS.py.

```
02142     def get_metapartition_extension(self, file_format):
02143         # get proper metapartition_extension string
02144         if file_format == "phylip":
02145             metapartition_extension = "-meta.phy"
02146         elif file_format == "phylip-int":
02147             metapartition_extension = "-meta.int-phy"
02148         elif file_format == "fasta":
02149             metapartition_extension = "-meta.fas"
02150         elif file_format == "nexus":
02151             metapartition_extension = "-meta.nex"
02152         elif file_format == "nexus-int":
02153             metapartition_extension = "-meta.int-nex"
02154 
02155         return metapartition_extension
02156 
```

Referenced by amas.AMAS.MetaAlignment.write_metapartitions().

Here is the caller graph for this function:


amas.AMAS.MetaAlignment.write_out
amas.AMAS.MetaAlignment.write_metapartitions
amas.AMAS.MetaAlignment.get_metapartition_extension

7.6.3.9 get_parsed_alignments()

```
amas.AMAS.MetaAlignment.get_parsed_alignments (
                self )
```

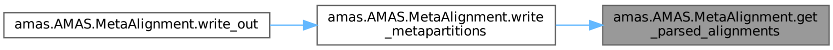
Definition at line 1543 of file AMAS.py.

```
    def get_parsed_alignments(self):
        # get parsed dictionaries with taxa and sequences
        parsed_alignments = []
        add_to_parsed_alignments = parsed_alignments.append
        for alignment in self.alignment_objects:
            parsed = alignment.parsed_aln
            add_to_parsed_alignments(parsed)
            # checking if every seq has the same length or if parsed is not empty; exit if false
            if self.check_align:
                equal = all(
                    x == [len(list(parsed.values())[i]) for i in range(0,len(list(parsed.values())))][0]
                    for x in [len(list(parsed.values())[i]) for i in range(0,len(list(parsed.values())))]
                )
                if equal is False:
                    print("ERROR: Sequences in input are of varying lengths. Be sure to align them first.")
                    sys.exit()

            if not parsed.keys() or not any(parsed.values()):
                print(
                    "ERROR: Parsed sequences of " + alignment.in_file + " are empty. "
                    "Are you sure you specified the right input format and/or that input is a valid alignment?"
                )
                sys.exit()

        return parsed_alignments

```

References amas.AMAS.MetaAlignment.alignment_objects, and amas.AMAS.MetaAlignment.check_align.

Referenced by amas.AMAS.MetaAlignment.write_metapartitions().

Here is the caller graph for this function:


amas.AMAS.MetaAlignment.write_out
amas.AMAS.MetaAlignment.write_metapartitions
amas.AMAS.MetaAlignment.get_parsed_alignments

7.6.3.10 get_partitioned()

```
amas.AMAS.MetaAlignment.get_partitioned (
                self,
                partitions_file )
```

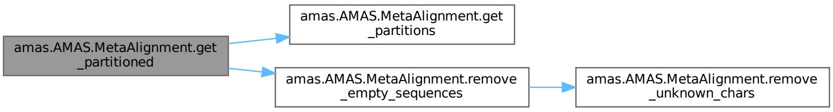
Definition at line 1569 of file AMAS.py.

```
    def get_partitioned(self, partitions_file):
        # partition alignment according to a partitions file
        partitions = self.get_partitions(partitions_file)
        alignment = self.parsed_alignments[0]

        # initiate list of newly partitioned alignments
        list_of_parts = []
        add_to_list_of_parts = list_of_parts.append
        for partition in partitions:
            # loop over all parsed partitions, adding taxa and sliced sequences
            for name, elements in partition.items():
                new_dict = {}

                for taxon, seq in alignment.items():
                    new_seq = ""

                    for dictionary in elements:
                        new_seq = new_seq +
seq[dictionary["start"]:dictionary["stop"]:dictionary["stride"]]
                        new_dict[taxon] = new_seq

                if self.remove_empty:
                    # check if remove empty sequences
                    no_empty_dict = self.remove_empty_sequences(new_dict)
                    add_to_list_of_parts({name : no_empty_dict})
                else:
                    # add partition name : dict of taxa and sequences to the list
                    add_to_list_of_parts({name : new_dict})

        return list_of_parts

```

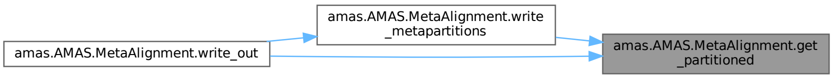
References amas.AMAS.MetaAlignment.get_partitions(), amas.AMAS.MetaAlignment.parsed_alignments, amas.AMAS.MetaAlignment.remove_empty, and amas.AMAS.MetaAlignment.remove_empty_sequences().

Referenced by amas.AMAS.MetaAlignment.write_metapartitions(), and amas.AMAS.MetaAlignment.write_out().

Here is the call graph for this function:


amas.AMAS.MetaAlignment.get
_partitioned
amas.AMAS.MetaAlignment.get
_partitions
amas.AMAS.MetaAlignment.remove
_empty_sequences
amas.AMAS.MetaAlignment.remove
_unknown_chars

Here is the caller graph for this function:


amas.AMAS.MetaAlignment.write_out
amas.AMAS.MetaAlignment.write
_metapartitions
amas.AMAS.MetaAlignment.get
_partitioned

7.6.3.11 get_partitions()

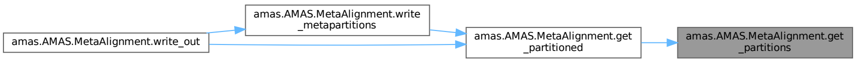
```
amas.AMAS.MetaAlignment.get_partitions (
                self,
                partitions_file )
```

Definition at line 1516 of file AMAS.py.

```
01516     def get_partitions(self, partitions_file):
01517         # parse and get partitions from partitions file
01518         partitions = FileParser(partitions_file)
01519         parsed_partitions = partitions.partitions_parse()
01520
01521         return parsed_partitions
01522
```

Referenced by amas.AMAS.MetaAlignment.get_partitioned().

Here is the caller graph for this function:


amas.AMAS.MetaAlignment.write_out
amas.AMAS.MetaAlignment.write
_metapartitions
amas.AMAS.MetaAlignment.get
_partitioned
amas.AMAS.MetaAlignment.get
_partitions

7.6.3.12 get_replicate()

```
amas.AMAS.MetaAlignment.get_replicate (
              self,
              no_replicates,
              no_loci )
```

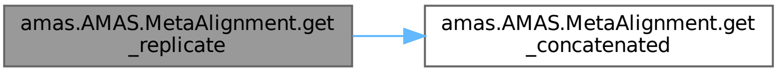
Definition at line 1726 of file AMAS.py.

```
    def get_replicate(self, no_replicates, no_loci):
        # construct replicate data sets for phylogenetic jackknife
        replicates = []
        add_to_replicates = replicates.append
        counter = 1
        for replicate in range(no_replicates):

            try:
                random_alignments = sample(self.parsed_alignments, no_loci)
            except ValueError:
                print("ERROR: You specified more loci per replicate than there are in your input.")
                sys.exit()

            random_alignments = sample(self.parsed_alignments, no_loci)
            concat_replicate = self.get_concatenated(random_alignments)[0]
            add_to_replicates(concat_replicate)
            counter += 1

        return replicates

```

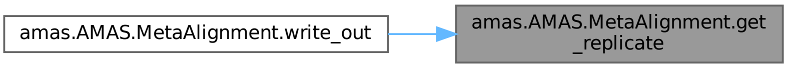
References amas.AMAS.MetaAlignment.get_concatenated(), and amas.AMAS.MetaAlignment.parsed_alignments.

Referenced by amas.AMAS.MetaAlignment.write_out().

Here is the call graph for this function:


amas.AMAS.MetaAlignment.get
_replicate
amas.AMAS.MetaAlignment.get
_concatenated

Here is the caller graph for this function:


amas.AMAS.MetaAlignment.write_out
amas.AMAS.MetaAlignment.get
_replicate

7.6.3.13 get_summaries()

```
amas.AMAS.MetaAlignment.get_summaries (
              self )
```

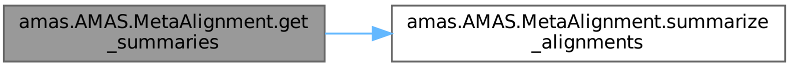
Definition at line 1599 of file AMAS.py.

```
    def get_summaries(self):
        # get summaries for all alignment objects

        # define different headers for aa and dna alignments
        aa_header = [
            "Alignment_name",
            "No_of_taxa",
            "Alignment_length",
            "Total_matrix_cells",
            "Undetermined_characters",
            "Missing_percent",
            "No_variable_sites",
            "Proportion_variable_sites",
            "Parsimony_informative_sites",
            "Proportion_parsimony_informative"
        ]

        dna_header = [
            "Alignment_name",
            "No_of_taxa",
            "Alignment_length",
            "Total_matrix_cells",
            "Undetermined_characters",
            "Missing_percent",
            "No_variable_sites",
            "Proportion_variable_sites",
            "Parsimony_informative_sites",
            "Proportion_parsimony_informative",
            "AT_content",
            "GC_content"
        ]

        alignments = self.alignment_objects
        parsed_alignments = self.parsed_alignments
        freq_header = [char for char in alignments[0].alphabet]

        if self.data_type == "aa":
            header = aa_header + freq_header
        elif self.data_type == "dna":
            header = dna_header + freq_header

        # use multiprocessing if more than one core specified
        if int(self.cores) == 1:
            summaries = [alignment.get_summary() for alignment in alignments]
        elif int(self.cores) > 1:
            pool = mp.Pool(int(self.cores))
            summaries = pool.map(self.summarize_alignments, alignments)
        return header, summaries
```

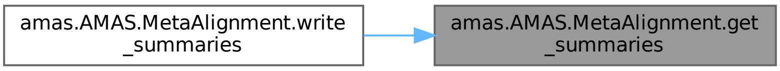
References amas.AMAS.MetaAlignment.alignment_objects, amas.AMAS.MetaAlignment.cores, amas.AMAS.Alignment.data_type, amas.AMAS.MetaAlignment.data_type, amas.AMAS.MetaAlignment.parsed_alignments, and amas.AMAS.MetaAlignment.summarize

Referenced by amas.AMAS.MetaAlignment.write_summaries().

Here is the call graph for this function:


amas.AMAS.MetaAlignment.get
_summaries
amas.AMAS.MetaAlignment.summarize
_alignments

Here is the caller graph for this function:


amas.AMAS.MetaAlignment.write
_summaries
amas.AMAS.MetaAlignment.get
_summaries

7.6.3.14 get_taxon_summaries()

```
amas.AMAS.MetaAlignment.get_taxon_summaries (
                self )
```

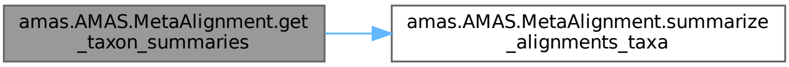
Definition at line 1653 of file AMAS.py.

```
01653     def get_taxon_summaries(self):
01654         # get per-sequence summaries for all alignment objects
01655 
01656         # define different headers for aa and dna alignments
01657         aa_header = [
01658             "Alignment_name",
01659             "Taxon_name",
01660             "Sequence_length",
01661             "Undetermined_characters",
01662             "Missing_percent"
01663         ]
01664 
01665         dna_header = [
01666             "Alignment_name",
01667             "Taxon_name",
01668             "Sequence_length",
01669             "Undetermined_characters",
01670             "Missing_percent",
01671             "AT_content",
01672             "GC_content"
01673         ]
01674 
01675         alignments = self.alignment_objects
01676         parsed_alignments = self.parsed_alignments
01677         freq_header = alignments[0].alphabet
01678 
01679         if self.data_type == "aa":
01680             header = aa_header + freq_header
01681         elif self.data_type == "dna":
01682             header = dna_header + freq_header
01683 
01684         # use multiprocessing if more than one core specified
01685         if int(self.cores) == 1:
01686             summaries = [alignment.get_taxa_summary() for alignment in alignments]
01687         elif int(self.cores) > 1:
01688             pool = mp.Pool(int(self.cores))
01689             summaries = pool.map(self.summarize_alignments_taxa, alignments)
01690 
01691         return header, summaries
01692 
```

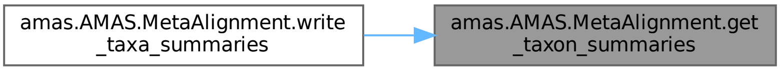
References amas.AMAS.MetaAlignment.alignment_objects, amas.AMAS.MetaAlignment.cores, amas.AMAS.Alignment.data_type, amas.AMAS.MetaAlignment.data_type, amas.AMAS.MetaAlignment.parsed_alignments, and amas.AMAS.MetaAlignment.summarize

Referenced by amas.AMAS.MetaAlignment.write_taxa_summaries().

Here is the call graph for this function:


amas.AMAS.MetaAlignment.get
_taxon_summaries
amas.AMAS.MetaAlignment.summarize
_alignments_taxa

Here is the caller graph for this function:


amas.AMAS.MetaAlignment.write
_taxa_summaries
amas.AMAS.MetaAlignment.get
_taxon_summaries

7.6.3.15 get_translated()

```
amas.AMAS.MetaAlignment.get_translated (
            self,
            translation_table,
            reading_frame )
```

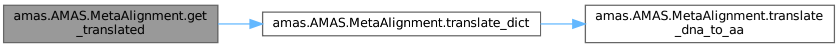
Definition at line 1478 of file AMAS.py.

```
01478     def get_translated(self, translation_table, reading_frame):
01479         if int(self.cores) == 1:
01480             translated_alignments = [self.translate_dict(alignment) for alignment in
      self.parsed_alignments]
01481         elif int(self.cores) > 1:
01482             pool = mp.Pool(int(self.cores))
01483             translated_alignments = pool.map(self.translate_dict, self.parsed_alignments)
01484 
01485         return translated_alignments
01486 
```

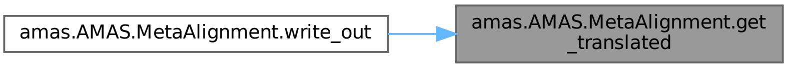
References amas.AMAS.MetaAlignment.cores, amas.AMAS.MetaAlignment.parsed_alignments, and amas.AMAS.MetaAlignment.tra

Referenced by amas.AMAS.MetaAlignment.write_out().

Here is the call graph for this function:


amas.AMAS.MetaAlignment.get
_translated
amas.AMAS.MetaAlignment.translate_dict
amas.AMAS.MetaAlignment.translate
_dna_to_aa

Here is the caller graph for this function:


amas.AMAS.MetaAlignment.write_out
amas.AMAS.MetaAlignment.get
_translated

7.6.3.16 get_trimmed()

```
amas.AMAS.MetaAlignment.get_trimmed (
            self,
            trim_fraction,
            parsimony_check )
```

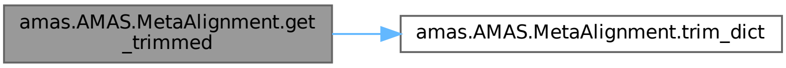
Definition at line 1495 of file AMAS.py.

```
01495     def get_trimmed(self, trim_fraction, parsimony_check):
01496         if int(self.cores) == 1:
01497             trimmed_alignments = [self.trim_dict(alignment) for alignment in self.alignment_objects]
01498         elif int(self.cores) > 1:
01499             pool = mp.Pool(int(self.cores))
01500             trimmed_alignments = pool.map(self.trim_dict, self.alignment_objects)
01501 
01502         return trimmed_alignments
01503 
```

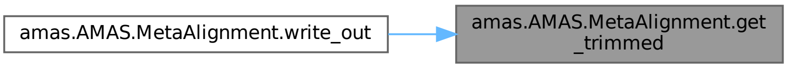
References amas.AMAS.MetaAlignment.alignment_objects, amas.AMAS.MetaAlignment.cores, and amas.AMAS.MetaAlignment.trim

Referenced by amas.AMAS.MetaAlignment.write_out().

Here is the call graph for this function:


amas.AMAS.MetaAlignment.get
_trimmed
amas.AMAS.MetaAlignment.trim_dict

Here is the caller graph for this function:


amas.AMAS.MetaAlignment.write_out
amas.AMAS.MetaAlignment.get
_trimmed

7.6.3.17 natural_sort()

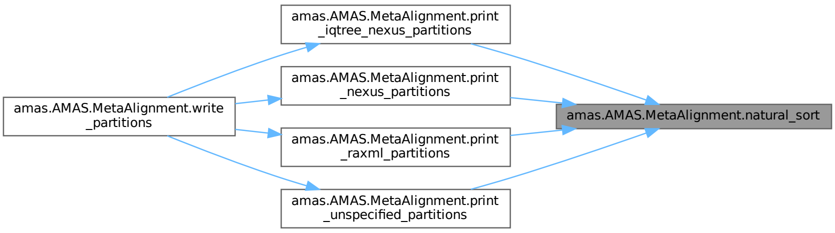
```
amas.AMAS.MetaAlignment.natural_sort (
                self,
                a_list )
```

Definition at line 1960 of file AMAS.py.

```
01960     def natural_sort(self, a_list):
01961         # create a function that does 'human sort' on a list
01962         convert = lambda text: int(text) if text.isdigit() else text.lower()
01963         alphanum_key = lambda key: [convert(c) for c in re.split('([0-9]+)', key)]
01964         return sorted(a_list, key = alphanum_key)
01965
```

Referenced by amas.AMAS.MetaAlignment.print_iqtree_nexus_partitions(), amas.AMAS.MetaAlignment.print_nexus_partitions(), amas.AMAS.MetaAlignment.print_raxml_partitions(), and amas.AMAS.MetaAlignment.print_unspecified_partitions().

Here is the caller graph for this function:


amas.AMAS.MetaAlignment.write
_partitions
amas.AMAS.MetaAlignment.print
_iqtree_nexus_partitions
amas.AMAS.MetaAlignment.print
_nexus_partitions
amas.AMAS.MetaAlignment.print
_raxml_partitions
amas.AMAS.MetaAlignment.print
_unspecified_partitions
amas.AMAS.MetaAlignment.natural_sort

7.6.3.18 print_fasta()

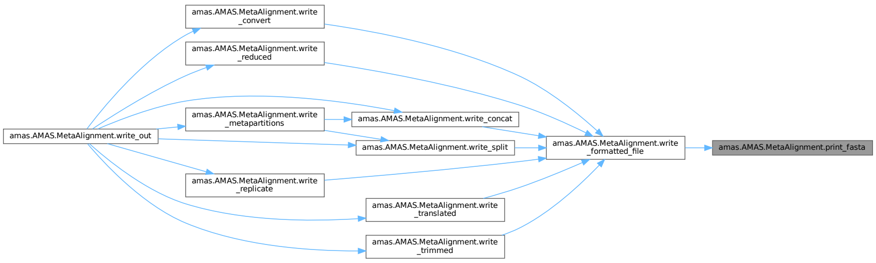
```
amas.AMAS.MetaAlignment.print_fasta (
                self,
                source_dict )
```

Definition at line 1833 of file AMAS.py.

```
    def print_fasta(self, source_dict):
        # print fasta-formatted string from a dictionary
        fasta_string = ""
        # each sequence line will have 80 characters
        n = 80

        for taxon, seq in sorted(source_dict.items()):
            # split dictionary values to a list of string, each n chars long
            seq = [seq[i:i+n] for i in range(0, len(seq), n)]
            # in case there are unwanted spaces in taxon names
            taxon = taxon.replace(" ", "_").strip("'")
            fasta_string += ">" + taxon + "\n"
            for element in seq:
                fasta_string += element + "\n"

        return fasta_string

```

Referenced by amas.AMAS.MetaAlignment.write_formatted_file().

Here is the caller graph for this function:


amas.AMAS.MetaAlignment.write_out
amas.AMAS.MetaAlignment.write _convert
amas.AMAS.MetaAlignment.write _reduced
amas.AMAS.MetaAlignment.write _metapartitions
amas.AMAS.MetaAlignment.write_concat
amas.AMAS.MetaAlignment.write_split
amas.AMAS.MetaAlignment.write _replicate
amas.AMAS.MetaAlignment.write _translated
amas.AMAS.MetaAlignment.write _trimmed
amas.AMAS.MetaAlignment.write _formatted_file
amas.AMAS.MetaAlignment.print_fasta

7.6.3.19 print_iqtree_nexus_partitions()

```
amas.AMAS.MetaAlignment.print_iqtree_nexus_partitions (
                self,
                data_type,
                codons )
```

Definition at line 2027 of file AMAS.py.

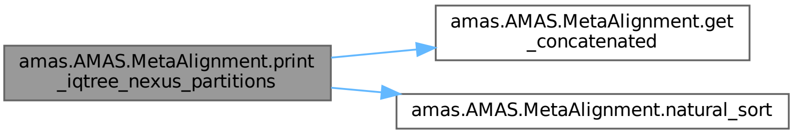
```
    def print_iqtree_nexus_partitions(self, data_type, codons):
        # print partitions for concatenated alignment
        part_string = ""
        part_dict = self.get_concatenated(self.parsed_alignments)[1]
        part_list = self.natural_sort(part_dict.keys())
        # write beginning of nexus sets
        part_string += "#nexus\n"
        part_string += "begin sets;\n"

        if data_type == "dna":
            if codons == "none":
                for key in part_list:
                    part_string += "  charset " + key + " = " + str(part_dict[key]) + ";\n"
            elif codons == "12":
                for key in part_list:
                    start, end = str(part_dict[key]).split("-")
                    part_string += "  charset " + key + "_pos1" + " = " + start + " - " + end + "\\2"
       + ";\n"
```

```
02044                     part_string += "  charset " + key + "_pos2" + " = " + str(int(start) + 1) + " - "
      + end + "\\2" + ";\n"
02045             elif codons == "123":
02046                 for key in part_list:
02047                     start, end = str(part_dict[key]).split("-")
02048                     part_string += "  charset " + key + "_pos1" + " = " + start + " - " + end + "\\3"
      + ";\n"
02049                     part_string += "  charset " + key + "_pos2" + " = " + str(int(start) + 1) + " - "
      + end + "\\3" + ";\n"
02050                     part_string += "  charset " + key + "_pos3" + " = " + str(int(start) + 2) + " - "
      + end + "\\3" + ";\n"
02051             part_string += "end;\n"
02052 
02053         elif data_type == "aa":
02054             for key in part_list:
02055                 part_string += "  charset " + key + " = " + str(part_dict[key]) + ";\n"
02056             part_string += "end;\n"
02057 
02058         return part_string
02059 
```

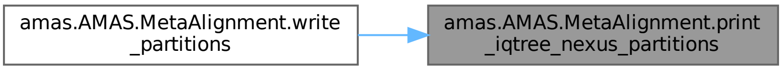
References amas.AMAS.MetaAlignment.get_concatenated(), amas.AMAS.MetaAlignment.natural_sort(), amas.AMAS.MetaAlignmen and amas.AMAS.MetaAlignment.split.

Referenced by amas.AMAS.MetaAlignment.write_partitions().

Here is the call graph for this function:


amas.AMAS.MetaAlignment.print
_iqtree_nexus_partitions
amas.AMAS.MetaAlignment.get
_concatenated
amas.AMAS.MetaAlignment.natural_sort

Here is the caller graph for this function:


amas.AMAS.MetaAlignment.write
_partitions
amas.AMAS.MetaAlignment.print
_iqtree_nexus_partitions

7.6.3.20 print_nexus()

```
amas.AMAS.MetaAlignment.print_nexus (
                self,
                source_dict )
```

Definition at line 1897 of file AMAS.py.

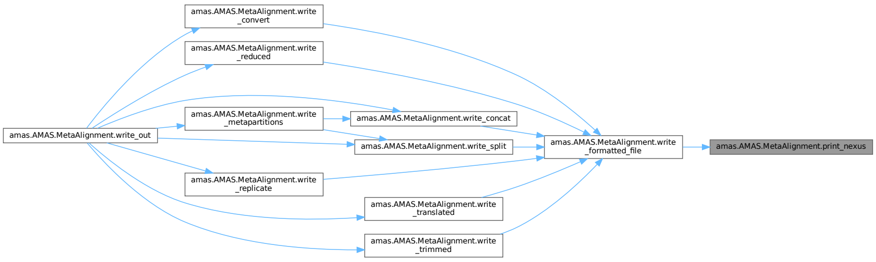
```
01897     def print_nexus(self, source_dict):
01898         # print nexus-formatted string from a dictionary
01899         if self.data_type == "aa" or self.command == "translate":
01900             data_type = "PROTEIN"
01901         elif self.data_type == "dna":
01902             data_type = "DNA"
01903 
```

```
        taxa_list = list(source_dict.keys())
        no_taxa = len(taxa_list)
        pad_longest_name = len(max(taxa_list, key=len)) + 3
        seq_length = len(next(iter(source_dict.values())))
        header = str(len(source_dict)) + " " + str(seq_length)
        nexus_string = (
            "#NEXUS\n\nBEGIN DATA;\n\tDIMENSIONS  NTAX=" + str(no_taxa) + " NCHAR=" + str(seq_length)
            + ";\n\tFORMAT DATATYPE=" + data_type + "  GAP = - MISSING = ?;\n\tMATRIX\n"
        )

        for taxon, seq in sorted(source_dict.items()):
            taxon = taxon.replace(" ", "_").strip("'")
            nexus_string += "\t" + taxon.ljust(pad_longest_name, ' ') + seq + "\n"
        nexus_string += "\n;\n\nEND;"

        return nexus_string

```

References amas.AMAS.MetaAlignment.command, amas.AMAS.Alignment.data_type, and amas.AMAS.MetaAlignment.data_type.

Referenced by amas.AMAS.MetaAlignment.write_formatted_file().

Here is the caller graph for this function:


amas.AMAS.MetaAlignment.write_out
amas.AMAS.MetaAlignment.write _convert
amas.AMAS.MetaAlignment.write _reduced
amas.AMAS.MetaAlignment.write _metapartitions
amas.AMAS.MetaAlignment.write _replicate
amas.AMAS.MetaAlignment.write_concat
amas.AMAS.MetaAlignment.write_split
amas.AMAS.MetaAlignment.write _translated
amas.AMAS.MetaAlignment.write _trimmed
amas.AMAS.MetaAlignment.write _formatted_file
amas.AMAS.MetaAlignment.print_nexus

7.6.3.21 print_nexus_int()

```
amas.AMAS.MetaAlignment.print_nexus_int (
                self,
                source_dict )
```

Definition at line 1921 of file AMAS.py.

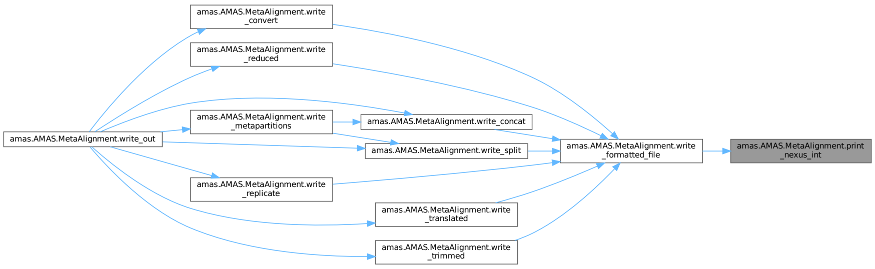
```
    def print_nexus_int(self, source_dict):
        # print nexus interleaved-formatted string from a dictionary
        if self.data_type == "aa":
            data_type = "PROTEIN"
        elif self.data_type == "dna":
            data_type = "DNA"

        taxa_list = list(source_dict.keys())
        no_taxa = len(taxa_list)
        pad_longest_name = len(max(taxa_list, key=len)) + 3
        seq_length = len(next(iter(source_dict.values())))
        header = str(len(source_dict)) + " " + str(seq_length)
        # this will be a list of tuples to hold taxa names and sequences
        seq_matrix = []
        nexus_int_string = (
            "#NEXUS\n\nBEGIN DATA;\n\tDIMENSIONS  NTAX=" + str(no_taxa) + " NCHAR=" + str(seq_length)
            + ";\n\tFORMAT   INTERLEAVE" + "   DATATYPE=" + data_type + "  GAP = - MISSING = ?;\n\tMATRIX\n"
        )
        # each sequence line will have 500 characters
        n = 500

        # recreate sequence matrix
        add_to_matrix = seq_matrix.append
        for taxon, seq in sorted(source_dict.items()):
            add_to_matrix((taxon, [seq[i:i+n] for i in range(0, len(seq), n)]))

```

```
01947         first_seq = seq_matrix[0][1]
01948         for index, item in enumerate(first_seq):
01949             for taxon, sequence in seq_matrix:
01950                 if index == 0:
01951                     nexus_int_string += taxon.ljust(pad_longest_name, ' ') + sequence[index] + "\n"
01952                 else:
01953                     nexus_int_string += sequence[index] + "\n"
01954             nexus_int_string += "\n"
01955 
01956         nexus_int_string += "\n;\n\nEND;"
01957 
01958         return nexus_int_string
01959 
```

References amas.AMAS.Alignment.data_type, and amas.AMAS.MetaAlignment.data_type.

Referenced by amas.AMAS.MetaAlignment.write_formatted_file().

Here is the caller graph for this function:


amas.AMAS.MetaAlignment.write_convert
amas.AMAS.MetaAlignment.write_reduced
amas.AMAS.MetaAlignment.write_metapartitions
amas.AMAS.MetaAlignment.write_concat
amas.AMAS.MetaAlignment.write_out
amas.AMAS.MetaAlignment.write_split
amas.AMAS.MetaAlignment.write_formatted_file
amas.AMAS.MetaAlignment.print_nexus_int
amas.AMAS.MetaAlignment.write_replicate
amas.AMAS.MetaAlignment.write_translated
amas.AMAS.MetaAlignment.write_trimmed

7.6.3.22 print_nexus_partitions()

```
amas.AMAS.MetaAlignment.print_nexus_partitions (
                self,
                data_type,
                codons )
```

Definition at line 1994 of file AMAS.py.

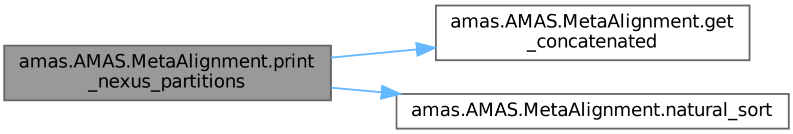
```
01994     def print_nexus_partitions(self, data_type, codons):
01995         # print partitions for concatenated alignment
01996         part_string = ""
01997         part_dict = self.get_concatenated(self.parsed_alignments)[1]
01998         part_list = self.natural_sort(part_dict.keys())
01999         # write beginning of nexus sets
02000         part_string += "#NEXUS\n\n"
02001         part_string += "BEGIN SETS;\n"
02002 
02003         if data_type == "dna":
02004             if codons == "none":
02005                 for key in part_list:
02006                     part_string += "\tcharset " + key + " = " + str(part_dict[key]) + ";\n"
02007             elif codons == "12":
02008                 for key in part_list:
02009                     start, end = str(part_dict[key]).split("-")
02010                     part_string += "\tcharset " + key + "_pos1" + " = " + start + "-" + end + "\\2" +
      ";\n"
02011                     part_string += "\tcharset " + key + "_pos2" + " = " + str(int(start) + 1) + "-" +
      end + "\\2" + ";\n"
02012             elif codons == "123":
02013                 for key in part_list:
02014                     start, end = str(part_dict[key]).split("-")
02015                     part_string += "\tcharset " + key + "_pos1" + " = " + start + "-" + end + "\\3" +
      ";\n"
02016                     part_string += "\tcharset " + key + "_pos2" + " = " + str(int(start) + 1) + "-" +
      end + "\\3" + ";\n"
02017                     part_string += "\tcharset " + key + "_pos3" + " = " + str(int(start) + 2) + "-" +
      end + "\\3" + ";\n"
02018             part_string += "END;"
```

```
02019
02020        elif data_type == "aa":
02021            for key in part_list:
02022                part_string += "\tcharset " + key + " = " + str(part_dict[key]) + ";\n"
02023            part_string += "END;"
02024
02025        return part_string
02026
```

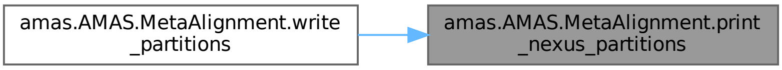
References amas.AMAS.MetaAlignment.get_concatenated(), amas.AMAS.MetaAlignment.natural_sort(), amas.AMAS.MetaAlignmen and amas.AMAS.MetaAlignment.split.

Referenced by amas.AMAS.MetaAlignment.write_partitions().

Here is the call graph for this function:


amas.AMAS.MetaAlignment.print
_nexus_partitions
amas.AMAS.MetaAlignment.get
_concatenated
amas.AMAS.MetaAlignment.natural_sort

Here is the caller graph for this function:


amas.AMAS.MetaAlignment.write
_partitions
amas.AMAS.MetaAlignment.print
_nexus_partitions

7.6.3.23 print_phylip()

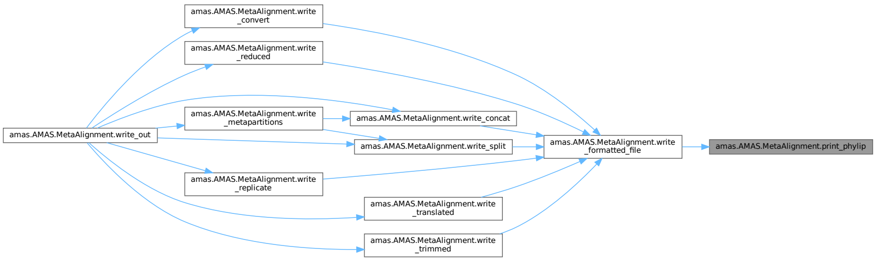
```
amas.AMAS.MetaAlignment.print_phylip (
                self,
                source_dict )
```

Definition at line 1850 of file AMAS.py.

```
01850    def print_phylip(self, source_dict):
01851        # print phylip-formatted string from a dictionary
01852        taxa_list = list(source_dict.keys())
01853        no_taxa = len(taxa_list)
01854        # figure out the max length of a taxon for nice padding of sequences
01855        pad_longest_name = len(max(taxa_list, key=len)) + 3
01856        # get sequence length from a random value
01857        seq_length = len(next(iter(source_dict.values())))
01858        header = str(len(source_dict)) + " " + str(seq_length)
01859        phylip_string = header + "\n"
01860        for taxon, seq in sorted(source_dict.items()):
01861            taxon = taxon.replace(" ", "_").strip("’")
01862            # left-justify taxon names relative to sequences
01863            phylip_string += taxon.ljust(pad_longest_name, ’ ’) + seq + "\n"
01864
01865        return phylip_string
01866
```

Referenced by amas.AMAS.MetaAlignment.write_formatted_file().

Here is the caller graph for this function:


amas.AMAS.MetaAlignment.write_convert
amas.AMAS.MetaAlignment.write_reduced
amas.AMAS.MetaAlignment.write_metapartitions
amas.AMAS.MetaAlignment.write_out
amas.AMAS.MetaAlignment.write_concat
amas.AMAS.MetaAlignment.write_split
amas.AMAS.MetaAlignment.write_formatted_file
amas.AMAS.MetaAlignment.print_phylip
amas.AMAS.MetaAlignment.write_replicate
amas.AMAS.MetaAlignment.write_translated
amas.AMAS.MetaAlignment.write_trimmed

7.6.3.24 print_phylip_int()

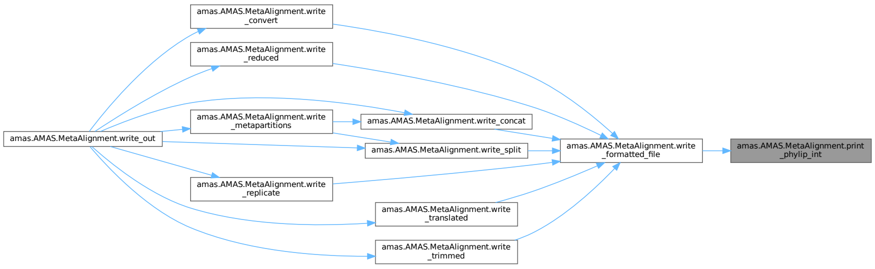
```
amas.AMAS.MetaAlignment.print_phylip_int (
                self,
                source_dict )
```

Definition at line 1867 of file AMAS.py.

```
01867     def print_phylip_int(self, source_dict):
01868         # print phylip interleaved-formatted string from a dictionary
01869         taxa_list = list(source_dict.keys())
01870         no_taxa = len(taxa_list)
01871         pad_longest_name = len(max(taxa_list, key=len)) + 3
01872         seq_length = len(next(iter(source_dict.values())))
01873         header = str(len(source_dict)) + " " + str(seq_length)
01874         phylip_int_string = header + "\n\n"
01875         # this will be a list of tuples to hold taxa names and sequences
01876         seq_matrix = []
01877 
01878         # each sequence line will have 500 characters
01879         n = 500
01880 
01881         # recreate sequence matrix
01882         add_to_matrix = seq_matrix.append
01883         for taxon, seq in sorted(source_dict.items()):
01884             add_to_matrix((taxon, [seq[i:i+n] for i in range(0, len(seq), n)]))
01885 
01886         first_seq = seq_matrix[0][1]
01887         for index, item in enumerate(first_seq):
01888             for taxon, sequence in seq_matrix:
01889                 if index == 0:
01890                     phylip_int_string += taxon.ljust(pad_longest_name, ’ ’) + sequence[index] + "\n"
01891                 else:
01892                     phylip_int_string += sequence[index] + "\n"
01893             phylip_int_string += "\n"
01894 
01895         return phylip_int_string
01896 
```

Referenced by amas.AMAS.MetaAlignment.write_formatted_file().

Here is the caller graph for this function:


amas.AMAS.MetaAlignment.write_convert
amas.AMAS.MetaAlignment.write_reduced
amas.AMAS.MetaAlignment.write_metapartitions
amas.AMAS.MetaAlignment.write_out
amas.AMAS.MetaAlignment.write_concat
amas.AMAS.MetaAlignment.write_split
amas.AMAS.MetaAlignment.write_formatted_file
amas.AMAS.MetaAlignment.print_phylip_int
amas.AMAS.MetaAlignment.write_replicate
amas.AMAS.MetaAlignment.write_translated
amas.AMAS.MetaAlignment.write_trimmed

7.6.3.25 print_raxml_partitions()

```
amas.AMAS.MetaAlignment.print_raxml_partitions (
                self,
                data_type,
                codons )
```

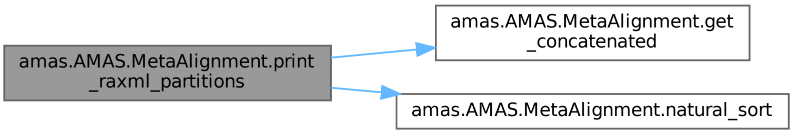
Definition at line 2060 of file AMAS.py.

```
02060     def print_raxml_partitions(self, data_type, codons):
02061         # print partitions for concatenated alignment
02062         part_string = ""
02063         part_dict = self.get_concatenated(self.parsed_alignments)[1]
02064         part_list = self.natural_sort(part_dict.keys())
02065 
02066         if data_type == "dna":
02067             if codons == "none":
02068                 for key in part_list:
02069                     part_string += "DNA, " + key + " = " + str(part_dict[key]) + "\n"
02070             elif codons == "12":
02071                 for key in part_list:
02072                     start, end = str(part_dict[key]).split("-")
02073                     part_string += "DNA, " + key + "_pos1" + " = " + start + "-" + end + "\\2" + "\n"
02074                     part_string += "DNA, " + key + "_pos2" + " = " + str(int(start) + 1) + "-" + end + "\\2" + "\n"
02075             elif codons == "123":
02076                 for key in part_list:
02077                     start, end = str(part_dict[key]).split("-")
02078                     part_string += "DNA, " + key + "_pos1" + " = " + start + "-" + end + "\\3" + "\n"
02079                     part_string += "DNA, " + key + "_pos2" + " = " + str(int(start) + 1) + "-" + end + "\\3" + "\n"
02080                     part_string += "DNA, " + key + "_pos3" + " = " + str(int(start) + 2) + "-" + end + "\\3" + "\n"
02081 
02082         elif data_type == "aa":
02083             for key in part_list:
02084                 part_string += "WAG, " + key + " = " + str(part_dict[key]) + "\n"
02085 
02086             #  aa-partition files with strides are probably not useful? (original below)
02087 #            elif codons == "12":
02088 #                for key in part_list:
02089 #                    start, end = str(part_dict[key]).split("-")
02090 #                    part_string += "WAG, " + key + "_pos1" + " = " + start + "-" + end + "\\2" + "\n"
02091 #                    part_string += "WAG, " + key + "_pos2" + " = " + str(int(start) + 1) + "-" + end + "\\2" + "\n"
02092 #            elif codons == "123":
02093 #                for key in part_list:
02094 #                    start, end = str(part_dict[key]).split("-")
02095 #                    part_string += "WAG, " + key + "_pos1" + " = " + start + "-" + end + "\\3" + "\n"
02096 #                    part_string += "WAG, " + key + "_pos2" + " = " + str(int(start) + 1) + "-" + end + "\\3" + "\n"
02097 #                    part_string += "WAG, " + key + "_pos3" + " = " + str(int(start) + 2) + "-" + end + "\\3" + "\n"
02098         return part_string
02099 
```

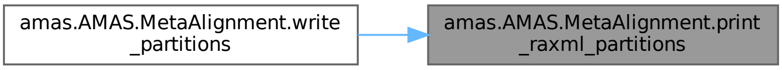
References amas.AMAS.MetaAlignment.get_concatenated(), amas.AMAS.MetaAlignment.natural_sort(), amas.AMAS.MetaAlignmen and amas.AMAS.MetaAlignment.split.

Referenced by amas.AMAS.MetaAlignment.write_partitions().

Here is the call graph for this function:


amas.AMAS.MetaAlignment.print
_raxml_partitions
amas.AMAS.MetaAlignment.get
_concatenated
amas.AMAS.MetaAlignment.natural_sort

Here is the caller graph for this function:


amas.AMAS.MetaAlignment.write
_partitions
amas.AMAS.MetaAlignment.print
_raxml_partitions

7.6.3.26 print_unspecified_partitions()

```
amas.AMAS.MetaAlignment.print_unspecified_partitions (
                self,
                data_type,
                codons )
```

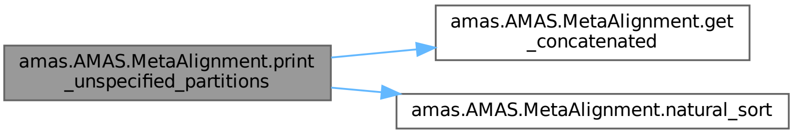
Definition at line 1966 of file AMAS.py.

```
01966     def print_unspecified_partitions(self, data_type, codons):
01967         # print partitions for concatenated alignment
01968         part_string = ""
01969         part_dict = self.get_concatenated(self.parsed_alignments)[1]
01970         part_list = self.natural_sort(part_dict.keys())
01971 
01972         if data_type == "dna":
01973             if codons == "none":
01974                 for key in part_list:
01975                     part_string += key + " = " + str(part_dict[key]) + "\n"
01976             elif codons == "12":
01977                 for key in part_list:
01978                     start, end = str(part_dict[key]).split("-")
01979                     part_string += key + "_pos1" + " = " + start + "-" + end + "\\2" + "\n"
01980                     part_string += key + "_pos2" + " = " + str(int(start) + 1) + "-" + end + "\\2" +
      "\n"
01981             elif codons == "123":
01982                 for key in part_list:
01983                     start, end = str(part_dict[key]).split("-")
01984                     part_string += key + "_pos1" + " = " + start + "-" + end + "\\3" + "\n"
01985                     part_string += key + "_pos2" + " = " + str(int(start) + 1) + "-" + end + "\\3" +
      "\n"
01986                     part_string += key + "_pos3" + " = " + str(int(start) + 2) + "-" + end + "\\3" +
      "\n"
01987 
01988         elif data_type == "aa":
01989             for key in part_list:
01990                 part_string += key + " = " + str(part_dict[key]) + "\n"
01991 
01992         return part_string
01993 
```

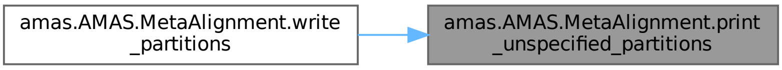
References amas.AMAS.MetaAlignment.get_concatenated(), amas.AMAS.MetaAlignment.natural_sort(), amas.AMAS.MetaAlignmen
and amas.AMAS.MetaAlignment.split.

Referenced by amas.AMAS.MetaAlignment.write_partitions().

Here is the call graph for this function:


amas.AMAS.MetaAlignment.print
_unspecified_partitions
amas.AMAS.MetaAlignment.get
_concatenated
amas.AMAS.MetaAlignment.natural_sort

Here is the caller graph for this function:


amas.AMAS.MetaAlignment.write
_partitions
amas.AMAS.MetaAlignment.print
_unspecified_partitions

7.6.3.27 remove_empty_sequences()

```
amas.AMAS.MetaAlignment.remove_empty_sequences (
                self,
                split_alignment )
```

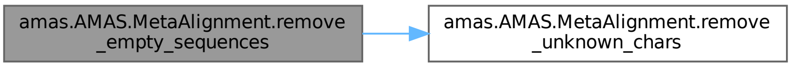
Definition at line 1510 of file AMAS.py.

```
01510     def remove_empty_sequences(self, split_alignment):
01511         # remove taxa from alignment if they are composed of only empty sequences
01512         new_alignment = {taxon : seq for taxon, seq in split_alignment.items() if
      self.remove_unknown_chars(seq)}
01513 
01514         return new_alignment
01515 
```

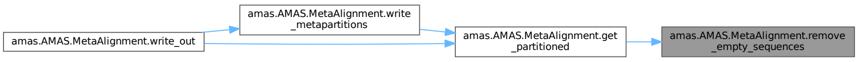
References amas.AMAS.MetaAlignment.remove_unknown_chars().

Referenced by amas.AMAS.MetaAlignment.get_partitioned().

Here is the call graph for this function:


amas.AMAS.MetaAlignment.remove
_empty_sequences
amas.AMAS.MetaAlignment.remove
_unknown_chars

Here is the caller graph for this function:


amas.AMAS.MetaAlignment.write_out
amas.AMAS.MetaAlignment.write
_metapartitions
amas.AMAS.MetaAlignment.get
_partitioned
amas.AMAS.MetaAlignment.remove
_empty_sequences

7.6.3.28 remove_from_alignment()

```
amas.AMAS.MetaAlignment.remove_from_alignment (
                self,
                alignment,
                species_to_remove_set,
                index )
```

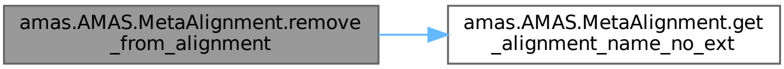
Definition at line 1807 of file AMAS.py.

```
01807     def remove_from_alignment(self, alignment, species_to_remove_set, index):
01808         # remove taxa from alignment
01809         aln_name = self.get_alignment_name_no_ext(index)
01810         for taxon in species_to_remove_set:
01811             if taxon not in alignment.keys():
01812                 print(
01813                     "WARNING: Taxon '" + taxon + "' not found in '" + aln_name + "'.\nIf you expected
      it to be there, "
01814                     "make sure to replace all taxon name spaces with underscores and that you are not
      using quotes."
01815                 )
01816         # originally within for-loop scope (redundancy)
01817         new_alignment = {species: seq for species, seq in alignment.items() if species not in
      species_to_remove_set}
01818
01819         return aln_name, new_alignment
01820
```

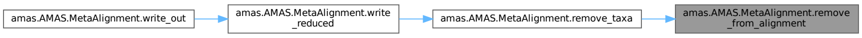
References amas.AMAS.MetaAlignment.get_alignment_name_no_ext().

Referenced by amas.AMAS.MetaAlignment.remove_taxa().

Here is the call graph for this function:


amas.AMAS.MetaAlignment.remove
_from_alignment
amas.AMAS.MetaAlignment.get
_alignment_name_no_ext

Here is the caller graph for this function:


amas.AMAS.MetaAlignment.write_out
amas.AMAS.MetaAlignment.write
_reduced
amas.AMAS.MetaAlignment.remove_taxa
amas.AMAS.MetaAlignment.remove
_from_alignment

7.6.3.29 remove_taxa()

```
amas.AMAS.MetaAlignment.remove_taxa (
                self,
                species_to_remove_set )
```

Definition at line 1821 of file AMAS.py.

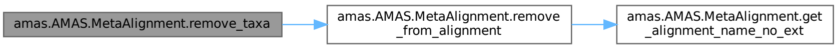
```
01821     def remove_taxa(self, species_to_remove_set):
01822         new_alns = {}
01823         for index, alignment in enumerate(self.parsed_alignments):
01824             aln_name, aln_dict = self.remove_from_alignment(alignment, species_to_remove_set, index)
01825             # check if alignment is not empty:
01826             if aln_dict:
```

```
                new_alns[aln_name] = aln_dict
            else:
                print("ERROR: You asked to remove all taxa from the alignment " + aln_name + ". No
      output file will be written.")

        return new_alns

```

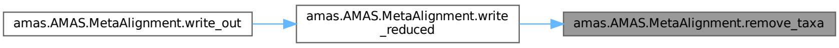
References amas.AMAS.MetaAlignment.parsed_alignments, and amas.AMAS.MetaAlignment.remove_from_alignment().

Referenced by amas.AMAS.MetaAlignment.write_reduced().

Here is the call graph for this function:


amas.AMAS.MetaAlignment.remove_taxa
amas.AMAS.MetaAlignment.remove_from_alignment
amas.AMAS.MetaAlignment.get_alignment_name_no_ext

Here is the caller graph for this function:


amas.AMAS.MetaAlignment.write_out
amas.AMAS.MetaAlignment.write_reduced
amas.AMAS.MetaAlignment.remove_taxa

7.6.3.30 remove_unknown_chars()

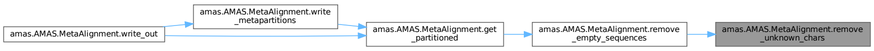
```
amas.AMAS.MetaAlignment.remove_unknown_chars (
                self,
                seq )
```

Definition at line 1504 of file AMAS.py.

```
01504     def remove_unknown_chars(self, seq):
01505         # remove unknown characters from sequence
01506         new_seq = seq.replace("?", "").replace("-", "")
01507
01508         return new_seq
01509
```

Referenced by amas.AMAS.MetaAlignment.remove_empty_sequences().

Here is the caller graph for this function:


amas.AMAS.MetaAlignment.write_out
amas.AMAS.MetaAlignment.write_metapartitions
amas.AMAS.MetaAlignment.get_partitioned
amas.AMAS.MetaAlignment.remove_empty_sequences
amas.AMAS.MetaAlignment.remove_unknown_chars

7.6.3.31 replace_string_in_file()

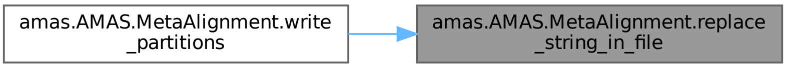
```
amas.AMAS.MetaAlignment.replace_string_in_file (
              self,
              file_name,
              old_string,
              new_string )
```

Definition at line 2100 of file AMAS.py.

```
02100     def replace_string_in_file(self, file_name, old_string, new_string):
02101         # global string replacement in file
02102         with open(file_name, "r", encoding="utf-8") as file:
02103             file_content = file.read()
02104         # write globally replaced content back to file
02105         glb_replaced_content = file_content.replace(old_string, new_string)
02106         with open(file_name, "w", encoding="utf-8") as file:
02107             file.write(glb_replaced_content)
02108 
```

Referenced by amas.AMAS.MetaAlignment.write_partitions().

Here is the caller graph for this function:


amas.AMAS.MetaAlignment.write
_partitions
amas.AMAS.MetaAlignment.replace
_string_in_file

7.6.3.32 summarize_alignments()

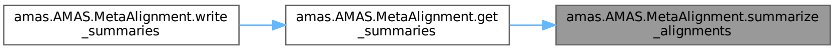
```
amas.AMAS.MetaAlignment.summarize_alignments (
              self,
              alignment )
```

Definition at line 1648 of file AMAS.py.

```
01648     def summarize_alignments(self, alignment):
01649         # helper function to summarize alignments
01650         summary = alignment.get_summary()
01651         return summary
01652 
```

Referenced by amas.AMAS.MetaAlignment.get_summaries().

Here is the caller graph for this function:


amas.AMAS.MetaAlignment.write
_summaries
amas.AMAS.MetaAlignment.get
_summaries
amas.AMAS.MetaAlignment.summarize
_alignments

7.6.3.33 summarize_alignments_taxa()

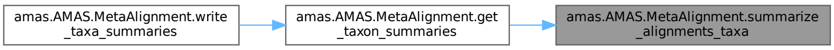
```
amas.AMAS.MetaAlignment.summarize_alignments_taxa (
              self,
              alignment )
```

Definition at line 1693 of file AMAS.py.

```
01693     def summarize_alignments_taxa(self, alignment):
01694         # helper function to summarize alignments by taxon
01695         summary = alignment.get_taxa_summary()
01696         return summary
01697
```

Referenced by amas.AMAS.MetaAlignment.get_taxon_summaries().

Here is the caller graph for this function:


amas.AMAS.MetaAlignment.write_taxa_summaries
amas.AMAS.MetaAlignment.get_taxon_summaries
amas.AMAS.MetaAlignment.summarize_alignments_taxa

7.6.3.34 translate_dict()

```
amas.AMAS.MetaAlignment.translate_dict (
              self,
              source_dict )
```

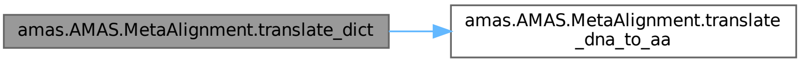
Definition at line 1467 of file AMAS.py.

```
01467     def translate_dict(self, source_dict):
01468         translation_table = self.codes.get(self.genetic_code)
01469         translated_dict = {}
01470         for taxon, seq in sorted(source_dict.items()):
01471             translated_seq = self.translate_dna_to_aa(seq, translation_table, self.reading_frame)
01472             if "*" in translated_seq:
01473                 print("WARNING: stop codon(s), indicated as *, found in {} sequence".format(taxon))
01474             translated_dict[taxon] = translated_seq
01475
01476         return translated_dict
01477
```

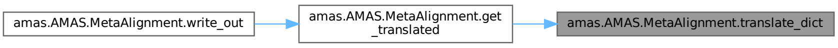
References amas.AMAS.MetaAlignment.codes, amas.AMAS.MetaAlignment.genetic_code, amas.AMAS.MetaAlignment.reading_fra and amas.AMAS.MetaAlignment.translate_dna_to_aa().

Referenced by amas.AMAS.MetaAlignment.get_translated().

Here is the call graph for this function:


amas.AMAS.MetaAlignment.translate_dict
amas.AMAS.MetaAlignment.translate_dna_to_aa

Here is the caller graph for this function:


amas.AMAS.MetaAlignment.write_out
amas.AMAS.MetaAlignment.get_translated
amas.AMAS.MetaAlignment.translate_dict

7.6.3.35 translate_dna_to_aa()

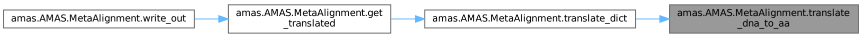
```
amas.AMAS.MetaAlignment.translate_dna_to_aa (
                self,
                seq,
                translation_table,
                frame )
```

Definition at line 1446 of file AMAS.py.

```
    def translate_dna_to_aa(self, seq, translation_table, frame):
        # translate DNA string into amino acids
        # where the last codon starts
        last_codon_start = len(seq) - 2
        # where the first codon starts
        if frame == 1:
            first = 0
        elif frame == 2:
            first = 1
        elif frame == 3:
            first = 2
        # create protein sequence by growing list
        protein = []
        add_to_protein = protein.append
        for start in range(first, last_codon_start, 3):
            codon = seq[start : start + 3]
            aa = translation_table.get(codon.upper(), ’X’)
            add_to_protein(aa)

        return "".join(protein)

```

Referenced by amas.AMAS.MetaAlignment.translate_dict().

Here is the caller graph for this function:


amas.AMAS.MetaAlignment.write_out
amas.AMAS.MetaAlignment.get_translated
amas.AMAS.MetaAlignment.translate_dict
amas.AMAS.MetaAlignment.translate_dna_to_aa

7.6.3.36 trim_dict()

```
amas.AMAS.MetaAlignment.trim_dict (
                self,
                alignment )
```

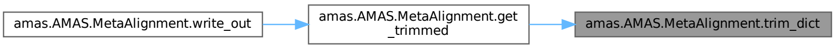
Definition at line 1487 of file AMAS.py.

```
    def trim_dict(self, alignment):
        trim_vector = alignment.get_trim_selection(self.trim_fraction, self.parsimony_check)
        aln_dict = alignment.parsed_aln
        for key in aln_dict:
            aln_dict[key] = ”.join(list(compress(aln_dict[key], trim_vector)))

        return aln_dict

```

References amas.AMAS.MetaAlignment.parsimony_check, and amas.AMAS.MetaAlignment.trim_fraction.

Referenced by amas.AMAS.MetaAlignment.get_trimmed().

Here is the caller graph for this function:


amas.AMAS.MetaAlignment.write_out
amas.AMAS.MetaAlignment.get_trimmed
amas.AMAS.MetaAlignment.trim_dict

7.6.3.37 write_concat()

```
amas.AMAS.MetaAlignment.write_concat (
                self,
                file_format )
```

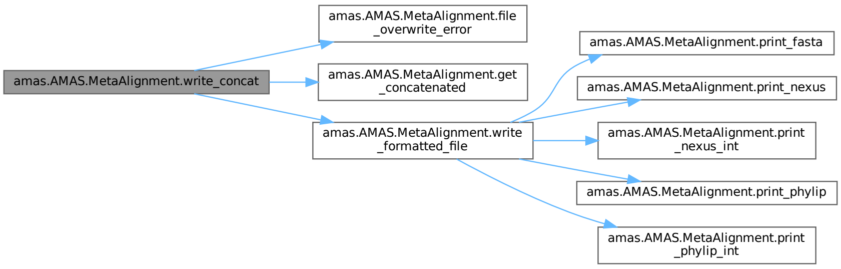
Definition at line 2188 of file AMAS.py.

```
02188     def write_concat(self, file_format):
02189         # write concatenated alignment into a file
02190         concatenated_alignment = self.get_concatenated(self.parsed_alignments)[0]
02191         file_name = self.concat_out
02192         self.file_overwrite_error(file_name)
02193         self.write_formatted_file(file_format, file_name, concatenated_alignment)
02194 
02195         print("Wrote concatenated sequences to " + file_format + " file '" + file_name + "'")
02196 
```

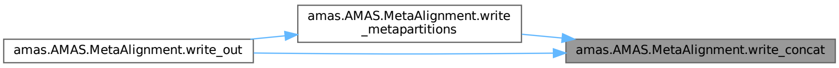
References amas.AMAS.MetaAlignment.concat_out, amas.AMAS.MetaAlignment.file_overwrite_error(), amas.AMAS.MetaAlignment. amas.AMAS.MetaAlignment.parsed_alignments, and amas.AMAS.MetaAlignment.write_formatted_file().

Referenced by amas.AMAS.MetaAlignment.write_metapartitions(), and amas.AMAS.MetaAlignment.write_out().

Here is the call graph for this function:


amas.AMAS.MetaAlignment.write_concat
amas.AMAS.MetaAlignment.file _overwrite_error
amas.AMAS.MetaAlignment.get _concatenated
amas.AMAS.MetaAlignment.write _formatted_file
amas.AMAS.MetaAlignment.print_fasta
amas.AMAS.MetaAlignment.print_nexus
amas.AMAS.MetaAlignment.print _nexus_int
amas.AMAS.MetaAlignment.print_phylip
amas.AMAS.MetaAlignment.print _phylip_int

Here is the caller graph for this function:


amas.AMAS.MetaAlignment.write_out
amas.AMAS.MetaAlignment.write _metapartitions
amas.AMAS.MetaAlignment.write_concat

7.6.3.38 write_convert()

```
amas.AMAS.MetaAlignment.write_convert (
                self,
                index,
                alignment,
                file_format,
                extension )
```

Definition at line 2197 of file AMAS.py.

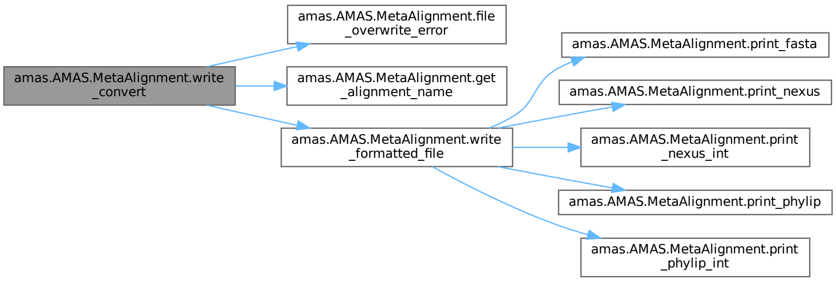
```
02197     def write_convert(self, index, alignment, file_format, extension):
02198         # write converted alignment into a file
```

```
        file_name = self.get_alignment_name(index, extension)
        self.file_overwrite_error(file_name)
        self.write_formatted_file(file_format, file_name, alignment)

```

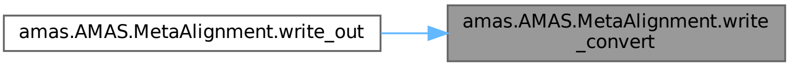
References amas.AMAS.MetaAlignment.file_overwrite_error(), amas.AMAS.MetaAlignment.get_alignment_name(), and amas.AMAS.MetaAlignment.write_formatted_file().

Referenced by amas.AMAS.MetaAlignment.write_out().

Here is the call graph for this function:


amas.AMAS.MetaAlignment.write_convert
amas.AMAS.MetaAlignment.file_overwrite_error
amas.AMAS.MetaAlignment.get_alignment_name
amas.AMAS.MetaAlignment.write_formatted_file
amas.AMAS.MetaAlignment.print_fasta
amas.AMAS.MetaAlignment.print_nexus
amas.AMAS.MetaAlignment.print_nexus_int
amas.AMAS.MetaAlignment.print_phylip
amas.AMAS.MetaAlignment.print_phylip_int

Here is the caller graph for this function:


amas.AMAS.MetaAlignment.write_out
amas.AMAS.MetaAlignment.write_convert

7.6.3.39 write_formatted_file()

```
amas.AMAS.MetaAlignment.write_formatted_file (
                self,
                file_format,
                file_name,
                alignment )
```

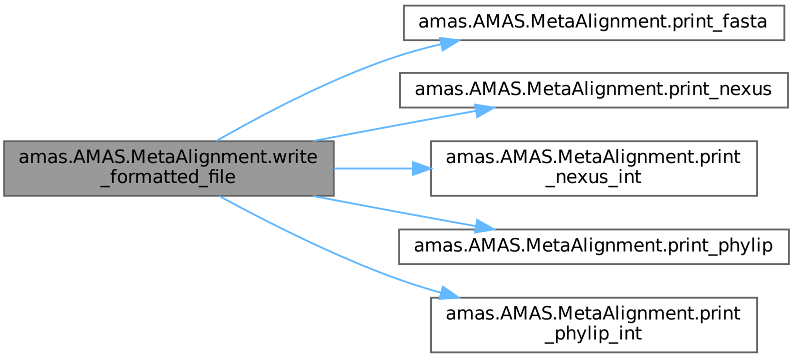
Definition at line 2162 of file AMAS.py.

```
02162     def write_formatted_file(self, file_format, file_name, alignment):
02163         # write the correct format string into a file
02164         with open(file_name, "w", encoding="utf-8") as out_file:
02165             if file_format == "phylip":
02166                 out_file.write(self.print_phylip(alignment))
02167             elif file_format == "fasta":
02168                 out_file.write(self.print_fasta(alignment))
02169             elif file_format == "phylip-int":
02170                 out_file.write(self.print_phylip_int(alignment))
02171             elif file_format == "nexus":
02172                 out_file.write(self.print_nexus(alignment))
02173             elif file_format == "nexus-int":
02174                 out_file.write(self.print_nexus_int(alignment))
02175
```

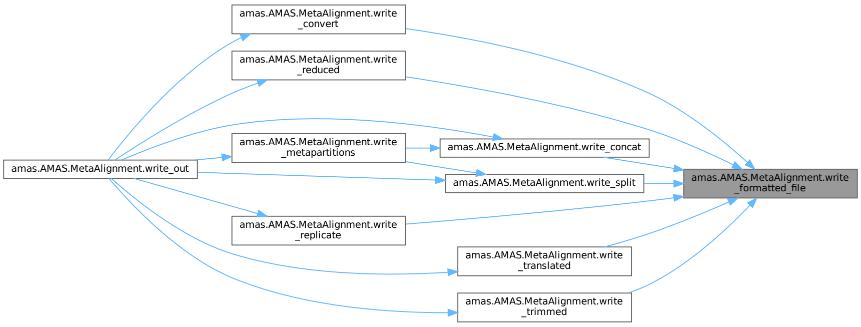
References amas.AMAS.MetaAlignment.print_fasta(), amas.AMAS.MetaAlignment.print_nexus(), amas.AMAS.MetaAlignment.print_r
amas.AMAS.MetaAlignment.print_phylip(), and amas.AMAS.MetaAlignment.print_phylip_int().

Referenced by amas.AMAS.MetaAlignment.write_concat(), amas.AMAS.MetaAlignment.write_convert(), amas.AMAS.MetaAlignment
amas.AMAS.MetaAlignment.write_replicate(), amas.AMAS.MetaAlignment.write_split(), amas.AMAS.MetaAlignment.write_translated
and amas.AMAS.MetaAlignment.write_trimmed().

Here is the call graph for this function:


amas.AMAS.MetaAlignment.write_formatted_file
amas.AMAS.MetaAlignment.print_fasta
amas.AMAS.MetaAlignment.print_nexus
amas.AMAS.MetaAlignment.print_nexus_int
amas.AMAS.MetaAlignment.print_phylip
amas.AMAS.MetaAlignment.print_phylip_int

Here is the caller graph for this function:


amas.AMAS.MetaAlignment.write_out
amas.AMAS.MetaAlignment.write_convert
amas.AMAS.MetaAlignment.write_reduced
amas.AMAS.MetaAlignment.write_metapartitions
amas.AMAS.MetaAlignment.write_concat
amas.AMAS.MetaAlignment.write_split
amas.AMAS.MetaAlignment.write_replicate
amas.AMAS.MetaAlignment.write_translated
amas.AMAS.MetaAlignment.write_trimmed
amas.AMAS.MetaAlignment.write_formatted_file

7.6.3.40 write_metapartitions()

```
amas.AMAS.MetaAlignment.write_metapartitions (
                self,
                file_format )
```

Definition at line 2263 of file AMAS.py.

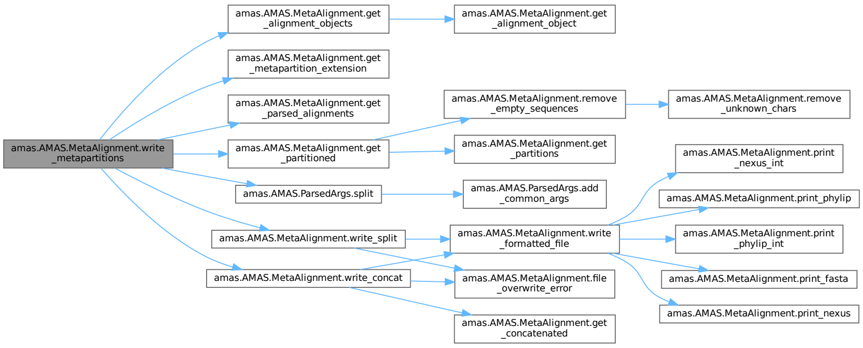
```
02263     def write_metapartitions(self, file_format):
02264         # write metapartitions - combines split and concat
02265         print("write_out elif action == metapartitions")
02266         metapartition_extension = self.get_metapartition_extension(file_format)
02267         list_of_alignments = self.get_partitioned(self.split)
02268         written_split_files = []
02269         err_indx = 0
```

```

        for item in list_of_alignments:
            try:
                for split_file in self.write_split(item, file_format, metapartition_extension):
                    written_split_files.append(split_file)
            except ValueError as e:
                    print("WARNING: ", e)
                    err_indx += 1
        if len(written_split_files) > 0:
            print("Wrote %d %s metapartition files from partitions provided" %
(len(written_split_files), file_format))
        if err_indx > 0:
            print("WARNING: %d file(s) raised an error while writing (see above)." % err_indx)

        # now set inputs to be the collated metapartition alignment files
        self.in_files = written_split_files
        self.alignment_objects = self.get_alignment_objects()
        self.parsed_alignments = self.get_parsed_alignments()

        # concat metapartition alignment files
        self.write_concat(file_format)

```

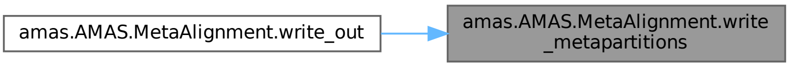
References amas.AMAS.MetaAlignment.alignment_objects, amas.AMAS.MetaAlignment.get_alignment_objects(), amas.AMAS.MetaAlignment.get_metapartition_extension(), amas.AMAS.MetaAlignment.get_parsed_alignments(), amas.AMAS.MetaAlignment.get_partitioned(), amas.AMAS.MetaAlignment.in_files, amas.AMAS.MetaAlignment.parsed_alignments, amas.AMAS.ParsedArgs.split(), amas.AMAS.MetaAlignment.split, amas.AMAS.MetaAlignment.write_concat(), and amas.AMAS.MetaAlignment.write_split().

Referenced by amas.AMAS.MetaAlignment.write_out().

Here is the call graph for this function:


amas.AMAS.MetaAlignment.write_metapartitions
amas.AMAS.MetaAlignment.get_alignment_objects
amas.AMAS.MetaAlignment.get_alignment_object
amas.AMAS.MetaAlignment.get_metapartition_extension
amas.AMAS.MetaAlignment.get_parsed_alignments
amas.AMAS.MetaAlignment.remove_empty_sequences
amas.AMAS.MetaAlignment.remove_unknown_chars
amas.AMAS.MetaAlignment.get_partitioned
amas.AMAS.MetaAlignment.get_partitions
amas.AMAS.MetaAlignment.print_nexus_int
amas.AMAS.ParsedArgs.split
amas.AMAS.ParsedArgs.add_common_args
amas.AMAS.MetaAlignment.print_phylip
amas.AMAS.MetaAlignment.write_split
amas.AMAS.MetaAlignment.write_formatted_file
amas.AMAS.MetaAlignment.print_phylip_int
amas.AMAS.MetaAlignment.write_concat
amas.AMAS.MetaAlignment.file_overwrite_error
amas.AMAS.MetaAlignment.print_fasta
amas.AMAS.MetaAlignment.print_nexus
amas.AMAS.MetaAlignment.get_concatenated

Here is the caller graph for this function:


amas.AMAS.MetaAlignment.write_out
amas.AMAS.MetaAlignment.write_metapartitions

7.6.3.41 write_out()

```
amas.AMAS.MetaAlignment.write_out (
              self,
```

action,

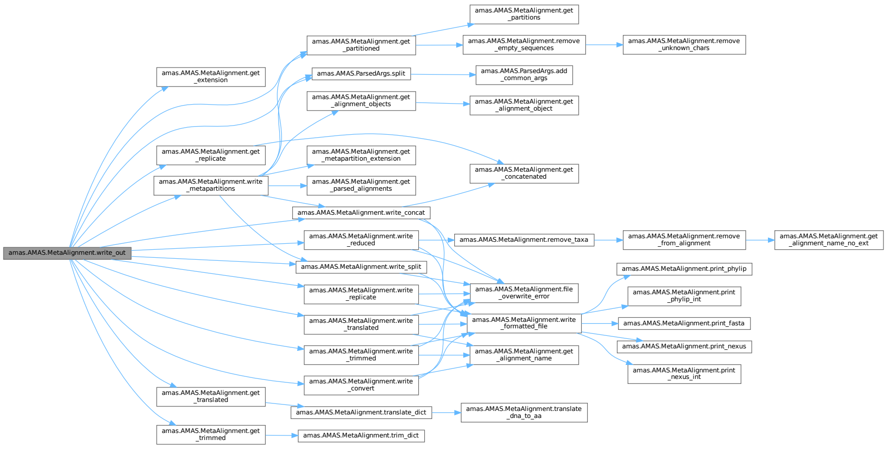
file_format)

Definition at line 2291 of file AMAS.py.

```
02291     def write_out(self, action, file_format):
02292         # write other output files depending on command (action)
02293         extension = self.get_extension(file_format)
02294 
02295         if action == "concat":
02296             self.write_concat(file_format)
02297 
02298         elif action == "convert":
02299             length = len(self.alignment_objects)
02300             [
02301                 self.write_convert(i, alignment, file_format, extension)
02302                 for i, alignment in enumerate(self.parsed_alignments)
02303             ]
02304             print("Converted " + str(length) + " files from " + self.in_format + " to " + file_format)
02305 
02306         elif action == "replicate":
02307             [
02308                 self.write_replicate(i, alignment, file_format, extension)
02309                 for i, alignment in enumerate(self.get_replicate(self.no_replicates, self.no_loci))
02310             ]
02311 
02312             print("Constructed " + str(self.no_replicates) + " replicate data sets, each from " +
      str(self.no_loci) + " alignments")
02313 
02314         elif action == "split":
02315             list_of_alignments = self.get_partitioned(self.split)
02316             written_split_files = []
02317             err_indx = 0
02318 
02319             for item in list_of_alignments:
02320                 try:
02321                     for split_file in self.write_split(item, file_format, extension):
02322                         written_split_files.append(split_file)
02323                 except ValueError as e:
02324                         print("WARNING: ", e)
02325                         err_indx += 1
02326             if len(written_split_files) > 0:
02327                 print("Wrote %d %s files from partitions provided" % (len(written_split_files),
      file_format))
02328             if err_indx > 0:
02329                 print("WARNING: %d file(s) raised an error while writing (see above)." % err_indx)
02330 
02331         elif action == "metapartitions":
02332             self.write_metapartitions(file_format)
02333 
02334         elif action == "remove":
02335             aln_no = self.write_reduced(file_format, extension)
02336             if aln_no:
02337                 print("Wrote " + str(aln_no) + " " + str(file_format) + " files with reduced taxon
      set")
02338 
02339         elif action == "translate":
02340             if self.data_type == "aa":
02341                 print("ERROR: cannot translate; you said your alignment already contains amino acids")
02342                 sys.exit()
02343             translated_alignment_dicts = self.get_translated(self.genetic_code, self.reading_frame)
02344             length = len(self.alignment_objects)
02345             [
02346                 self.write_translated(i, alignment, file_format, extension)
02347                 for i, alignment in enumerate(translated_alignment_dicts)
02348             ]
02349             print("Translated " + str(length) + " files to amino acid sequences")
02350 
02351         elif action == "trim": # self.trim_fraction, self.parsimony_check
02352             trimmed_alignment_dicts = self.get_trimmed(self.trim_fraction, self.parsimony_check)
02353             length = len(self.alignment_objects)
02354             [
02355                 self.write_trimmed(i, alignment, file_format, extension)
02356                 for i, alignment in enumerate(trimmed_alignment_dicts)
02357             ]
02358             print("Trimmed", str(length), "file(s) to have", self.trim_fraction, "minimum occupancy
      per alignment column")
02359 
02360 
```

References amas.AMAS.MetaAlignment.alignment_objects, amas.AMAS.Alignment.data_type, amas.AMAS.MetaAlignment.data_typ amas.AMAS.MetaAlignment.genetic_code, amas.AMAS.MetaAlignment.get_extension(), amas.AMAS.MetaAlignment.get_partitioned amas.AMAS.MetaAlignment.get_replicate(), amas.AMAS.MetaAlignment.get_translated(), amas.AMAS.MetaAlignment.get_trimmed(amas.AMAS.Alignment.in_format, amas.AMAS.MetaAlignment.in_format, amas.AMAS.MetaAlignment.no_loci, amas.AMAS.MetaAlignment.no_replicates, amas.AMAS.MetaAlignment.parsed_alignments, amas.AMAS.MetaAlignment.parsimony_ amas.AMAS.MetaAlignment.reading_frame, amas.AMAS.ParsedArgs.split(), amas.AMAS.MetaAlignment.split, amas.AMAS.MetaAlignment.trim_fraction, amas.AMAS.MetaAlignment.write_concat(), amas.AMAS.MetaAlignment.write_convert(), amas.AMAS.MetaAlignment.write_metapartitions(), amas.AMAS.MetaAlignment.write_reduced(), amas.AMAS.MetaAlignment.write_ amas.AMAS.MetaAlignment.write_split(), amas.AMAS.MetaAlignment.write_translated(), and amas.AMAS.MetaAlignment.write_trimi

Here is the call graph for this function:


amas.AMAS.MetaAlignment.write_out
amas.AMAS.MetaAlignment.get_extension
amas.AMAS.MetaAlignment.get_partitioned
amas.AMAS.ParsedArgs.split
amas.AMAS.MetaAlignment.get_alignment_objects
amas.AMAS.MetaAlignment.get_replicate
amas.AMAS.MetaAlignment.write_metapartitions
amas.AMAS.MetaAlignment.get_metapartition_extension
amas.AMAS.MetaAlignment.get_parsed_alignments
amas.AMAS.MetaAlignment.write_concat
amas.AMAS.MetaAlignment.write_reduced
amas.AMAS.MetaAlignment.write_split
amas.AMAS.MetaAlignment.write_replicate
amas.AMAS.MetaAlignment.write_translated
amas.AMAS.MetaAlignment.write_trimmed
amas.AMAS.MetaAlignment.write_convert
amas.AMAS.MetaAlignment.get_translated
amas.AMAS.MetaAlignment.translate_dict
amas.AMAS.MetaAlignment.get_trimmed
amas.AMAS.MetaAlignment.trim_dict
amas.AMAS.MetaAlignment.get_partitions
amas.AMAS.MetaAlignment.remove_empty_sequences
amas.AMAS.MetaAlignment.remove_unknown_chars
amas.AMAS.ParsedArgs.add_common_args
amas.AMAS.MetaAlignment.get_alignment_object
amas.AMAS.MetaAlignment.get_concatenated
amas.AMAS.MetaAlignment.remove_taxa
amas.AMAS.MetaAlignment.remove_from_alignment
amas.AMAS.MetaAlignment.get_alignment_name_no_ext
amas.AMAS.MetaAlignment.print_phylip
amas.AMAS.MetaAlignment.file_overwrite_error
amas.AMAS.MetaAlignment.print_phylip_int
amas.AMAS.MetaAlignment.write_formatted_file
amas.AMAS.MetaAlignment.print_fasta
amas.AMAS.MetaAlignment.print_nexus
amas.AMAS.MetaAlignment.get_alignment_name
amas.AMAS.MetaAlignment.print_nexus_int
amas.AMAS.MetaAlignment.translate_dna_to_aa

7.6.3.42 write_partitions()

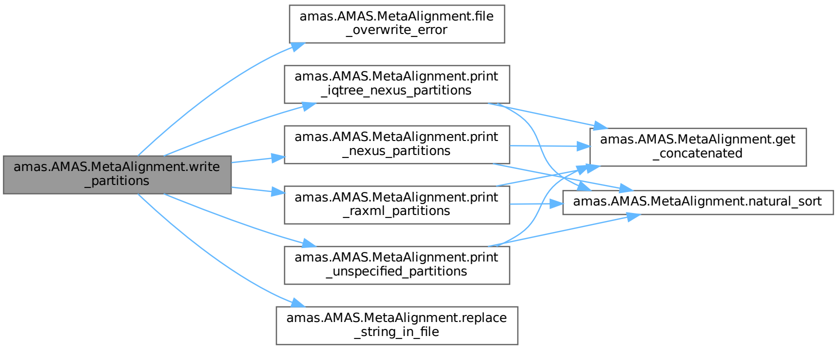
```
amas.AMAS.MetaAlignment.write_partitions (
              self,
              file_name,
              part_format,
              data_type,
              codons )
```

Definition at line 2109 of file AMAS.py.

```
02109     def write_partitions(self, file_name, part_format, data_type, codons):
02110         # write partitions file for concatenated alignment
02111         self.file_overwrite_error(file_name)
02112         with open(file_name, "w", encoding="utf-8") as part_file:
02113             if part_format == "nexus":
02114                 part_file.write(self.print_nexus_partitions(data_type, codons))
02115             if part_format == "iqtree-nexus":
02116                 part_file.write(self.print_iqtree_nexus_partitions(data_type, codons))
02117             if part_format == "raxml":
02118                 part_file.write(self.print_raxml_partitions(data_type, codons))
02119             if part_format == "unspecified":
02120                 part_file.write(self.print_unspecified_partitions(data_type, codons))
02121 
02122             if self.using_metapartitions:
02123                 self.replace_string_in_file(file_name, '-meta =', ' =')
02124 
02125         print("Wrote partitions for the concatenated file to '" + file_name + "'")
02126 
```

References amas.AMAS.MetaAlignment.file_overwrite_error(), amas.AMAS.MetaAlignment.print_iqtree_nexus_partitions(), amas.AMAS.MetaAlignment.print_nexus_partitions(), amas.AMAS.MetaAlignment.print_raxml_partitions(), amas.AMAS.MetaAlignment.print_unspecified_partitions(), amas.AMAS.MetaAlignment.replace_string_in_file(), and amas.AMAS.MetaAlignment.using_metapartitions.

Here is the call graph for this function:


amas.AMAS.MetaAlignment.write_partitions
amas.AMAS.MetaAlignment.file_overwrite_error
amas.AMAS.MetaAlignment.print_iqtree_nexus_partitions
amas.AMAS.MetaAlignment.print_nexus_partitions
amas.AMAS.MetaAlignment.print_raxml_partitions
amas.AMAS.MetaAlignment.print_unspecified_partitions
amas.AMAS.MetaAlignment.replace_string_in_file
amas.AMAS.MetaAlignment.get_concatenated
amas.AMAS.MetaAlignment.natural_sort

7.6.3.43 write_reduced()

```
amas.AMAS.MetaAlignment.write_reduced (
                self,
                file_format,
                extension )
```

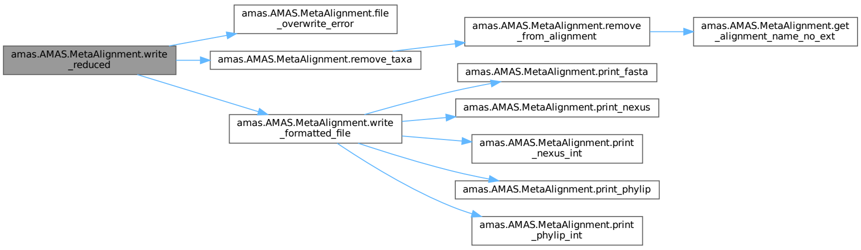
Definition at line 2234 of file AMAS.py.

```
02234     def write_reduced(self, file_format, extension):
02235         # write alignment with taxa removed into a file
02236         prefix = self.reduced_file_prefix
02237         alns = self.remove_taxa(self.species_to_remove)
02238         for file_name, aln_dict in alns.items():
02239             out_file_name = prefix + file_name + extension
02240             self.file_overwrite_error(out_file_name)
02241             self.write_formatted_file(file_format, out_file_name, aln_dict)
02242         return len(alns)
02243 
```

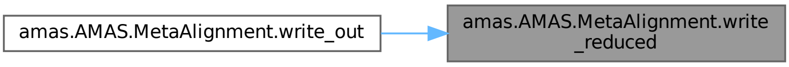
References amas.AMAS.MetaAlignment.file_overwrite_error(), amas.AMAS.MetaAlignment.reduced_file_prefix, amas.AMAS.MetaAlignment.remove_taxa(), amas.AMAS.MetaAlignment.species_to_remove, and amas.AMAS.MetaAlignment.write_

Referenced by amas.AMAS.MetaAlignment.write_out().

Here is the call graph for this function:


amas.AMAS.MetaAlignment.write_reduced
amas.AMAS.MetaAlignment.file_overwrite_error
amas.AMAS.MetaAlignment.remove_taxa
amas.AMAS.MetaAlignment.write_formatted_file
amas.AMAS.MetaAlignment.remove_from_alignment
amas.AMAS.MetaAlignment.get_alignment_name_no_ext
amas.AMAS.MetaAlignment.print_fasta
amas.AMAS.MetaAlignment.print_nexus
amas.AMAS.MetaAlignment.print_nexus_int
amas.AMAS.MetaAlignment.print_phylip
amas.AMAS.MetaAlignment.print_phylip_int

Here is the caller graph for this function:


amas.AMAS.MetaAlignment.write_out
amas.AMAS.MetaAlignment.write
_reduced

7.6.3.44 write_replicate()

```
amas.AMAS.MetaAlignment.write_replicate (
                self,
                index,
                alignment,
                file_format,
                extension )
```

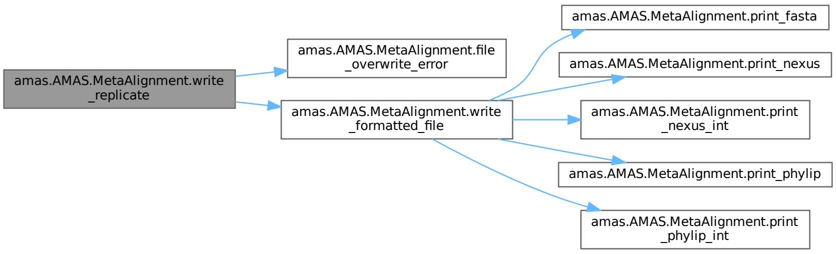
Definition at line 2203 of file AMAS.py.

```
02203     def write_replicate(self, index, alignment, file_format, extension):
02204         # write replicate alignment into a file
02205         file_name = "replicate" + str(index + 1) + "_" + str(self.no_loci) + "-loci" + extension
02206         self.file_overwrite_error(file_name)
02207         self.write_formatted_file(file_format, file_name, alignment)
02208
```

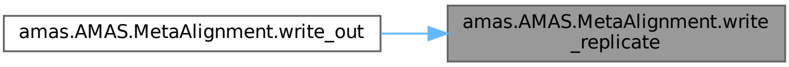
References amas.AMAS.MetaAlignment.file_overwrite_error(), amas.AMAS.MetaAlignment.no_loci, and amas.AMAS.MetaAlignment

Referenced by amas.AMAS.MetaAlignment.write_out().

Here is the call graph for this function:


amas.AMAS.MetaAlignment.write
_replicate
amas.AMAS.MetaAlignment.file
_overwrite_error
amas.AMAS.MetaAlignment.write
_formatted_file
amas.AMAS.MetaAlignment.print_fasta
amas.AMAS.MetaAlignment.print_nexus
amas.AMAS.MetaAlignment.print
_nexus_int
amas.AMAS.MetaAlignment.print_phylip
amas.AMAS.MetaAlignment.print
_phylip_int

Here is the caller graph for this function:


amas.AMAS.MetaAlignment.write_out
amas.AMAS.MetaAlignment.write
_replicate

7.6.3.45 write_split()

```
amas.AMAS.MetaAlignment.write_split (
                self,
                item,
                file_format,
                extension )
```

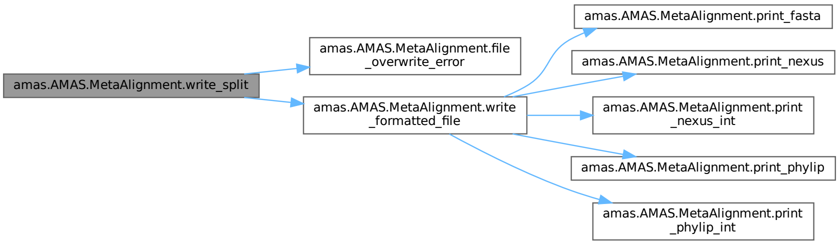
Definition at line 2209 of file AMAS.py.

```
02209     def write_split(self, item, file_format, extension):
02210         # write split alignments from partitions file
02211         # bad practice with the dicts; figure out better solution
02212         partition_name = list(item.keys())[0]
02213         alignment = item[partition_name]
02214 
02215         if not alignment:
02216             # If the alignment dict is empty, i.e. no alignment associated with partition name, raise error
02217             raise ValueError("Partition '%s' is empty. No sequences to write." % partition_name)
02218 
02219         # implementation of option --no-san (don't prepend input superalignment filename to the `split` outputs)
02220         if self.no_sup_aln_name:
02221             file_name = partition_name + extension
02222         else:
02223             file_name = str(self.in_files[0].split('.')[0]) + "_" + partition_name + extension
02224 
02225         try:
02226             self.file_overwrite_error(file_name)
02227             self.write_formatted_file(file_format, file_name, alignment)
02228             yield file_name
02229         except ValueError as e:
02230             print("There was an issue writing file '%s': %s" % (file_name, str(e)))
02231             remove(file_name)
02232             raise
02233 
```

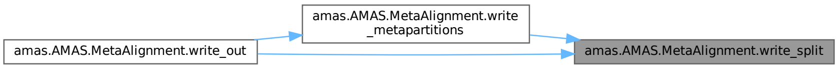
References amas.AMAS.MetaAlignment.file_overwrite_error(), amas.AMAS.MetaAlignment.in_files, amas.AMAS.MetaAlignment.no_ amas.AMAS.MetaAlignment.split, and amas.AMAS.MetaAlignment.write_formatted_file().

Referenced by amas.AMAS.MetaAlignment.write_metapartitions(), and amas.AMAS.MetaAlignment.write_out().

Here is the call graph for this function:


amas.AMAS.MetaAlignment.write_split
amas.AMAS.MetaAlignment.file_overwrite_error
amas.AMAS.MetaAlignment.write_formatted_file
amas.AMAS.MetaAlignment.print_fasta
amas.AMAS.MetaAlignment.print_nexus
amas.AMAS.MetaAlignment.print_nexus_int
amas.AMAS.MetaAlignment.print_phylip
amas.AMAS.MetaAlignment.print_phylip_int

Here is the caller graph for this function:


amas.AMAS.MetaAlignment.write_out
amas.AMAS.MetaAlignment.write_metapartitions
amas.AMAS.MetaAlignment.write_split

7.6.3.46 write_summaries()

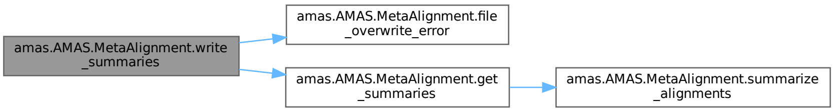
```
amas.AMAS.MetaAlignment.write_summaries (
                self,
                file_name )
```

Definition at line 1698 of file AMAS.py.

```
01698     def write_summaries(self, file_name):
01699         # write summaries to file
01700 
01701         self.file_overwrite_error(file_name)
01702 
01703         with open(file_name, "w", encoding="utf-8") as summary_file:
01704             summary_out = self.get_summaries()
01705             header = ’\t’.join(summary_out[0])
01706             new_summ = [’\t’.join(summary) for summary in summary_out[1]]
01707             summary_file.write(header + ’\n’)
01708             summary_file.write(’\n’.join(new_summ))
01709             summary_file.write(’\n’)
01710             print("Wrote summaries to file ’" + file_name + "’")
01711 
```

References amas.AMAS.MetaAlignment.file_overwrite_error(), and amas.AMAS.MetaAlignment.get_summaries().

Here is the call graph for this function:


amas.AMAS.MetaAlignment.write_summaries
amas.AMAS.MetaAlignment.file_overwrite_error
amas.AMAS.MetaAlignment.get_summaries
amas.AMAS.MetaAlignment.summarize_alignments

7.6.3.47 write_taxa_summaries()

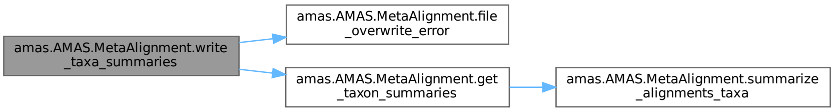
```
amas.AMAS.MetaAlignment.write_taxa_summaries (
                self )
```

Definition at line 1712 of file AMAS.py.

```
01712     def write_taxa_summaries(self):
01713         # write by-taxon summaries to file
01714         for index, in_file_name in enumerate(self.in_files):
01715             out_file_name = in_file_name + "-seq-summary.txt"
01716             self.file_overwrite_error(out_file_name)
01717             with open(out_file_name, "w", encoding="utf-8") as summary_file:
01718                 summary_out = self.get_taxon_summaries()
01719                 header = ’\t’.join(summary_out[0])
01720                 summ = [[str(col) for col in element] for element in summary_out[1][index]]
01721                 new_summ = [’\t’.join(row) for row in summ]
01722                 summary_file.write(header + ’\n’)
01723                 summary_file.write(’\n’.join(new_summ))
01724                 summary_file.write(’\n’)
01725 
```

References amas.AMAS.MetaAlignment.file_overwrite_error(), amas.AMAS.MetaAlignment.get_taxon_summaries(), and amas.AMAS.MetaAlignment.in_files.

Here is the call graph for this function:


amas.AMAS.MetaAlignment.write_taxa_summaries
amas.AMAS.MetaAlignment.file_overwrite_error
amas.AMAS.MetaAlignment.get_taxon_summaries
amas.AMAS.MetaAlignment.summarize_alignments_taxa

7.6.3.48 write_translated()

```
amas.AMAS.MetaAlignment.write_translated (
                self,
                index,
                alignment,
                file_format,
                extension )
```

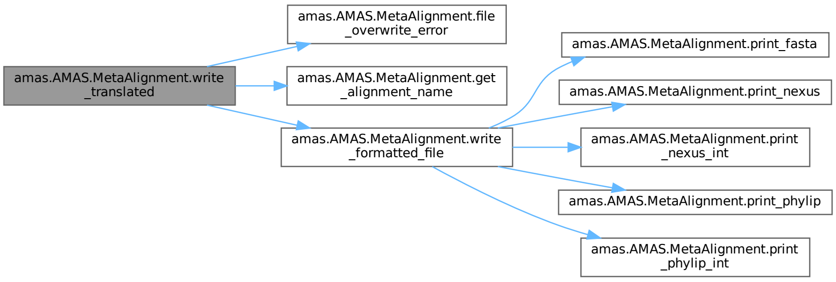
Definition at line 2244 of file AMAS.py.

```
02244     def write_translated(self, index, alignment, file_format, extension):
02245         # write alignments translated into amino acids
02246         prefix = "translated_"
02247         file_name = self.get_alignment_name(index, extension)
02248         out_file_name = prefix + file_name + extension
02249         self.file_overwrite_error(out_file_name)
02250         self.write_formatted_file(file_format, out_file_name, alignment)
02251
```

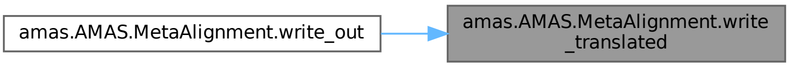
References amas.AMAS.MetaAlignment.file_overwrite_error(), amas.AMAS.MetaAlignment.get_alignment_name(), and amas.AMAS.MetaAlignment.write_formatted_file().

Referenced by amas.AMAS.MetaAlignment.write_out().

Here is the call graph for this function:


amas.AMAS.MetaAlignment.write_translated
amas.AMAS.MetaAlignment.file_overwrite_error
amas.AMAS.MetaAlignment.get_alignment_name
amas.AMAS.MetaAlignment.write_formatted_file
amas.AMAS.MetaAlignment.print_fasta
amas.AMAS.MetaAlignment.print_nexus
amas.AMAS.MetaAlignment.print_nexus_int
amas.AMAS.MetaAlignment.print_phylip
amas.AMAS.MetaAlignment.print_phylip_int

Here is the caller graph for this function:


amas.AMAS.MetaAlignment.write_out
amas.AMAS.MetaAlignment.write_translated

7.6.3.49 write_trimmed()

```
amas.AMAS.MetaAlignment.write_trimmed (
                self,
                index,
                alignment,
            file_format,
            extension )
```

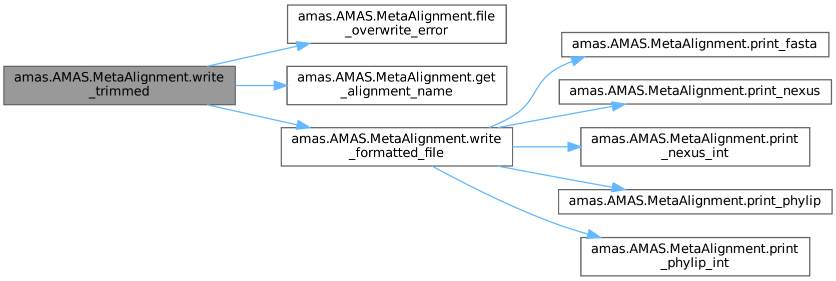
Definition at line 2252 of file AMAS.py.

```
02252     def write_trimmed(self, index, alignment, file_format, extension):
02253         # write trimmed alignments
02254         if self.trim_out:
02255             out_file_name = self.trim_out
02256         else:
02257             prefix = "trimmed_"
02258             file_name = self.get_alignment_name(index, extension)
02259             out_file_name = prefix + file_name
02260         self.file_overwrite_error(out_file_name)
02261         self.write_formatted_file(file_format, out_file_name, alignment)
02262 
```

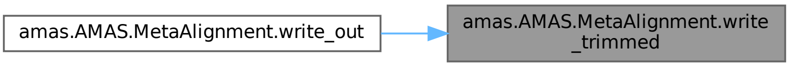
References amas.AMAS.MetaAlignment.file_overwrite_error(), amas.AMAS.MetaAlignment.get_alignment_name(), amas.AMAS.MetaAlignment.trim_out, and amas.AMAS.MetaAlignment.write_formatted_file().

Referenced by amas.AMAS.MetaAlignment.write_out().

Here is the call graph for this function:


amas.AMAS.MetaAlignment.write_trimmed
amas.AMAS.MetaAlignment.file_overwrite_error
amas.AMAS.MetaAlignment.get_alignment_name
amas.AMAS.MetaAlignment.write_formatted_file
amas.AMAS.MetaAlignment.print_fasta
amas.AMAS.MetaAlignment.print_nexus
amas.AMAS.MetaAlignment.print_nexus_int
amas.AMAS.MetaAlignment.print_phylip
amas.AMAS.MetaAlignment.print_phylip_int

Here is the caller graph for this function:


amas.AMAS.MetaAlignment.write_out
amas.AMAS.MetaAlignment.write_trimmed

7.6.4 Member Data Documentation

7.6.4.1 alignment_objects

`amas.AMAS.MetaAlignment.alignment_objects`

Definition at line 1224 of file AMAS.py.

Referenced by amas.AMAS.MetaAlignment.get_alignment_name(), amas.AMAS.MetaAlignment.get_alignment_name_no_ext(), amas.AMAS.MetaAlignment.get_concatenated(), amas.AMAS.MetaAlignment.get_parsed_alignments(), amas.AMAS.MetaAlignment amas.AMAS.MetaAlignment.get_taxon_summaries(), amas.AMAS.MetaAlignment.get_trimmed(), amas.AMAS.MetaAlignment.write_ and amas.AMAS.MetaAlignment.write_out().

7.6.4.2 by_taxon_summary

`amas.AMAS.MetaAlignment.by_taxon_summary`

Definition at line 1178 of file AMAS.py.

7.6.4.3 check_align

`amas.AMAS.MetaAlignment.check_align`

Definition at line 1176 of file AMAS.py.

Referenced by amas.AMAS.MetaAlignment.get_parsed_alignments().

7.6.4.4 check_taxa

`amas.AMAS.MetaAlignment.check_taxa`

Definition at line 1213 of file AMAS.py.

7.6.4.5 codes

`amas.AMAS.MetaAlignment.codes`

Definition at line 1424 of file AMAS.py.

Referenced by amas.AMAS.MetaAlignment.translate_dict().

7.6.4.6 codes_list

`amas.AMAS.MetaAlignment.codes_list`

Definition at line 1228 of file AMAS.py.

7.6.4.7 codons

`amas.AMAS.MetaAlignment.codons`

Definition at line 1183 of file AMAS.py.

7.6.4.8 command

`amas.AMAS.MetaAlignment.command`

Definition at line 1173 of file AMAS.py.

Referenced by amas.AMAS.MetaAlignment.print_nexus().

7.6.4.9 concat_out

`amas.AMAS.MetaAlignment.concat_out`

Definition at line 1174 of file AMAS.py.

Referenced by amas.AMAS.MetaAlignment.write_concat().

7.6.4.10 cores

`amas.AMAS.MetaAlignment.cores`

Definition at line 1177 of file AMAS.py.

Referenced by amas.AMAS.MetaAlignment.get_alignment_objects(), amas.AMAS.MetaAlignment.get_summaries(), amas.AMAS.MetaAlignment.get_taxon_summaries(), amas.AMAS.MetaAlignment.get_translated(), and amas.AMAS.MetaAlignment.

7.6.4.11 data_type

`amas.AMAS.MetaAlignment.data_type`

Definition at line 1172 of file AMAS.py.

Referenced by amas.AMAS.MetaAlignment.get_alignment_object(), amas.AMAS.MetaAlignment.get_summaries(), amas.AMAS.MetaAlignment.get_taxon_summaries(), amas.AMAS.MetaAlignment.print_nexus(), amas.AMAS.MetaAlignment.print_r and amas.AMAS.MetaAlignment.write_out().

7.6.4.12 gencode_NCBI_1

`amas.AMAS.MetaAlignment.gencode_NCBI_1`

Definition at line 1251 of file AMAS.py.

7.6.4.13 gencode_NCBI_10

`amas.AMAS.MetaAlignment.gencode_NCBI_10`

Definition at line 1364 of file AMAS.py.

7.6.4.14 gencode_NCBI_11

`amas.AMAS.MetaAlignment.gencode_NCBI_11`

Definition at line 1368 of file AMAS.py.

7.6.4.15 gencode_NCBI_12

`amas.AMAS.MetaAlignment.gencode_NCBI_12`

Definition at line 1371 of file AMAS.py.

7.6.4.16 gencode_NCBI_13

`amas.AMAS.MetaAlignment.gencode_NCBI_13`

Definition at line 1375 of file AMAS.py.

7.6.4.17 gencode_NCBI_14

`amas.AMAS.MetaAlignment.gencode_NCBI_14`

Definition at line 1382 of file AMAS.py.

7.6.4.18 gencode_NCBI_16

`amas.AMAS.MetaAlignment.gencode_NCBI_16`

Definition at line 1390 of file AMAS.py.

7.6.4.19 gencode_NCBI_2

`amas.AMAS.MetaAlignment.gencode_NCBI_2`

Definition at line 1322 of file AMAS.py.

7.6.4.20 gencode_NCBI_21

`amas.AMAS.MetaAlignment.gencode_NCBI_21`

Definition at line 1394 of file AMAS.py.

7.6.4.21 gencode_NCBI_22

`amas.AMAS.MetaAlignment.gencode_NCBI_22`

Definition at line 1402 of file AMAS.py.

7.6.4.22 gencode_NCBI_23

`amas.AMAS.MetaAlignment.gencode_NCBI_23`

Definition at line 1407 of file AMAS.py.

7.6.4.23 gencode_NCBI_24

`amas.AMAS.MetaAlignment.gencode_NCBI_24`

Definition at line 1411 of file AMAS.py.

7.6.4.24 gencode_NCBI_25

`amas.AMAS.MetaAlignment.gencode_NCBI_25`

Definition at line 1417 of file AMAS.py.

7.6.4.25 gencode_NCBI_26

`amas.AMAS.MetaAlignment.gencode_NCBI_26`

Definition at line 1421 of file AMAS.py.

7.6.4.26 gencode_NCBI_3

`amas.AMAS.MetaAlignment.gencode_NCBI_3`

Definition at line 1329 of file AMAS.py.

7.6.4.27 gencode_NCBI_4

`amas.AMAS.MetaAlignment.gencode_NCBI_4`

Definition at line 1341 of file AMAS.py.

7.6.4.28 gencode_NCBI_5

`amas.AMAS.MetaAlignment.gencode_NCBI_5`

Definition at line 1345 of file AMAS.py.

7.6.4.29 gencode_NCBI_6

`amas.AMAS.MetaAlignment.gencode_NCBI_6`

Definition at line 1352 of file AMAS.py.

7.6.4.30 gencode_NCBI_9

`amas.AMAS.MetaAlignment.gencode_NCBI_9`

Definition at line 1357 of file AMAS.py.

7.6.4.31 genetic_code

`amas.AMAS.MetaAlignment.genetic_code`

Definition at line 1217 of file AMAS.py.

Referenced by amas.AMAS.MetaAlignment.translate_dict(), and amas.AMAS.MetaAlignment.write_out().

7.6.4.32 in_files

`amas.AMAS.MetaAlignment.in_files`

Definition at line 1170 of file AMAS.py.

Referenced by amas.AMAS.MetaAlignment.get_alignment_objects(), amas.AMAS.MetaAlignment.write_metapartitions(), amas.AMAS.MetaAlignment.write_split(), and amas.AMAS.MetaAlignment.write_taxa_summaries().

7.6.4.33 in_format

`amas.AMAS.MetaAlignment.in_format`

Definition at line 1171 of file AMAS.py.

Referenced by amas.AMAS.MetaAlignment.get_alignment_object(), amas.AMAS.Alignment.get_parsed_aln(), and amas.AMAS.MetaAlignment.write_out().

7.6.4.34 no_loci

`amas.AMAS.MetaAlignment.no_loci`

Definition at line 1190 of file AMAS.py.

Referenced by amas.AMAS.MetaAlignment.write_out(), and amas.AMAS.MetaAlignment.write_replicate().

7.6.4.35 no_mpan

`amas.AMAS.MetaAlignment.no_mpan`

Definition at line 1180 of file AMAS.py.

Referenced by amas.AMAS.MetaAlignment.get_concatenated().

7.6.4.36 no_replicates

`amas.AMAS.MetaAlignment.no_replicates`

Definition at line 1189 of file AMAS.py.

Referenced by amas.AMAS.MetaAlignment.write_out().

7.6.4.37 no_sup_aln_name

`amas.AMAS.MetaAlignment.no_sup_aln_name`

Definition at line 1179 of file AMAS.py.

Referenced by amas.AMAS.MetaAlignment.write_split().

7.6.4.38 parsed_alignments

`amas.AMAS.MetaAlignment.parsed_alignments`

Definition at line 1225 of file AMAS.py.

Referenced by amas.AMAS.MetaAlignment.get_partitioned(), amas.AMAS.MetaAlignment.get_replicate(), amas.AMAS.MetaAlignment.get_summaries(), amas.AMAS.MetaAlignment.get_taxon_summaries(), amas.AMAS.MetaAlignment.get amas.AMAS.MetaAlignment.print_iqtree_nexus_partitions(), amas.AMAS.MetaAlignment.print_nexus_partitions(), amas.AMAS.MetaAlignment.print_raxml_partitions(), amas.AMAS.MetaAlignment.print_unspecified_partitions(), amas.AMAS.MetaAlignment.remove_taxa(), amas.AMAS.MetaAlignment.write_concat(), amas.AMAS.MetaAlignment.write_metapart and amas.AMAS.MetaAlignment.write_out().

7.6.4.39 parsimony_check

`amas.AMAS.MetaAlignment.parsimony_check`

Definition at line 1222 of file AMAS.py.

Referenced by amas.AMAS.MetaAlignment.trim_dict(), and amas.AMAS.MetaAlignment.write_out().

7.6.4.40 prepend_label

`amas.AMAS.MetaAlignment.prepend_label`

Definition at line 1203 of file AMAS.py.

Referenced by amas.AMAS.MetaAlignment.get_concatenated().

7.6.4.41 reading_frame

`amas.AMAS.MetaAlignment.reading_frame`

Definition at line 1216 of file AMAS.py.

Referenced by amas.AMAS.MetaAlignment.translate_dict(), and amas.AMAS.MetaAlignment.write_out().

7.6.4.42 reduced_file_prefix

`amas.AMAS.MetaAlignment.reduced_file_prefix`

Definition at line 1212 of file AMAS.py.

Referenced by amas.AMAS.MetaAlignment.write_reduced().

7.6.4.43 remove_empty

`amas.AMAS.MetaAlignment.remove_empty`

Definition at line 1194 of file AMAS.py.

Referenced by amas.AMAS.MetaAlignment.get_partitioned().

7.6.4.44 species_to_remove

`amas.AMAS.MetaAlignment.species_to_remove`

Definition at line 1210 of file AMAS.py.

Referenced by amas.AMAS.MetaAlignment.write_reduced().

7.6.4.45 species_to_remove_set

`amas.AMAS.MetaAlignment.species_to_remove_set`

Definition at line 1211 of file AMAS.py.

7.6.4.46 split

`amas.AMAS.MetaAlignment.split`

Definition at line 1193 of file AMAS.py.

Referenced by amas.AMAS.MetaAlignment.get_concatenated(), amas.AMAS.MetaAlignment.print_iqtree_nexus_partitions(), amas.AMAS.MetaAlignment.print_nexus_partitions(), amas.AMAS.MetaAlignment.print_raxml_partitions(), amas.AMAS.MetaAlignment.print_unspecified_partitions(), amas.AMAS.MetaAlignment.write_metapartitions(), amas.AMAS.MetaAlignment.write_out(), and amas.AMAS.MetaAlignment.write_split().

7.6.4.47 trim_fraction

`amas.AMAS.MetaAlignment.trim_fraction`

Definition at line 1220 of file AMAS.py.

Referenced by amas.AMAS.MetaAlignment.trim_dict(), and amas.AMAS.MetaAlignment.write_out().

7.6.4.48 trim_out

`amas.AMAS.MetaAlignment.trim_out`

Definition at line 1221 of file AMAS.py.

Referenced by amas.AMAS.MetaAlignment.write_trimmed().

7.6.4.49 using_metapartitions

`amas.AMAS.MetaAlignment.using_metapartitions`

Definition at line 1175 of file AMAS.py.

Referenced by amas.AMAS.MetaAlignment.get_concatenated(), and amas.AMAS.MetaAlignment.write_partitions().

The documentation for this class was generated from the following file:

- amas/AMAS.py

7.7 amas.AMAS.ParsedArgs Class Reference

Public Member Functions

- __init__ (self)
- add_common_args (self, parser)
- trim (self)
- summary (self)
- concat (self)
- convert (self)
- replicate (self)
- split (self)
- metapartitions (self)
- translate (self)
- remove (self)
- get_args_dict (self)

Public Attributes

- args

7.7.1 Detailed Description

Definition at line 50 of file AMAS.py.

7.7.2 Constructor & Destructor Documentation

7.7.2.1 __init__()

```
amas.AMAS.ParsedArgs.__init__ (
                self )
```

Definition at line 52 of file AMAS.py.

```
00052     def __init__(self):
00053         parser = argparse.ArgumentParser(
00054             usage=”’AMAS <command> [<args>]
00055 
00056 The AMAS commands are:
00057   concat            Concatenate input alignments.
00058   convert           Convert to other file format.
00059   replicate         Create replicate data sets for phylogenetic jackknife.
00060   split             Split alignment according to a partitions file.
00061   summary           Write alignment summary.
00062   remove            Remove taxa from alignment.
00063   translate         Translate DNA alignment into protein alignment.
00064   trim              Remove columns from alignment.
00065   metapartitions    Runs `split` and concatenates the output.
00066 
00067 
00068 Use AMAS <command> -h for help with arguments of the command of interest
00069 ”’
00070         )
00071 
00072         parser.add_argument(
00073             "command",
00074             help="Subcommand to run"
00075         )
00076 
00077         # parse_args defaults to [1:] for args, but you need to
00078         # exclude the rest of the args too, or validation will fail
00079         self.args = parser.parse_args(sys.argv[1:2])
00080         if not hasattr(self, self.args.command):
00081             print(’Unrecognized command’)
00082             parser.print_help()
00083             exit(1)
00084         # use dispatch pattern to invoke method with same name
00085         getattr(self, self.args.command)()
00086 
```

7.7.3 Member Function Documentation

7.7.3.1 add_common_args()

```
amas.AMAS.ParsedArgs.add_common_args (
                self,
                parser )
```

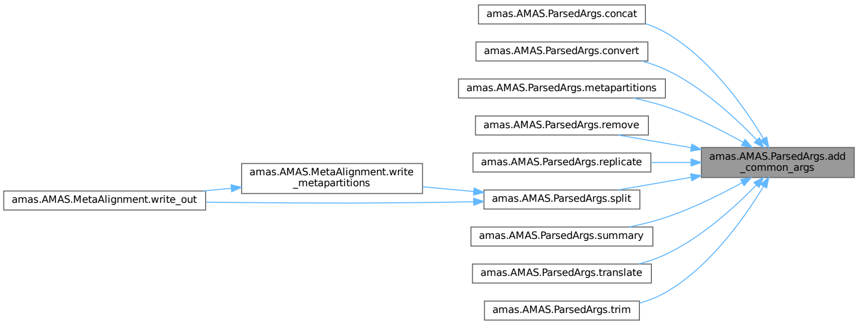
Definition at line 87 of file AMAS.py.

```
00087     def add_common_args(self, parser):
00088         # define required arguments for every command
00089         requiredNamed = parser.add_argument_group(’required arguments’)
00090         parser.add_argument(
00091             "-e",
00092             "--check-align",
00093             dest = "check_align",
00094             action = "store_true",
00095             default = False,
00096             help = "Check if input sequences are aligned. Default: no check"
00097         )
00098         parser.add_argument(
00099             # parallelization is used for file parsing and calculating summary stats
00100             "-c",
00101             "--cores",
00102             dest = "cores",
00103             default = 1,
00104             help = "Number of cores used. Default: 1"
00105         )
00106 
```

```
00107         requiredNamed.add_argument(
00108             "-i",
00109             "--in-files",
00110             nargs = "+",
00111             type = str,
00112             dest = "in_files",
00113             required = True,
00114             help = """Alignment files to be taken as input.
00115             You can specify multiple files using wildcards (e.g. --in-files *fasta)"""
00116         )
00117         requiredNamed.add_argument(
00118             "-f",
00119             "--in-format",
00120             dest = "in_format",
00121             required = True,
00122             choices = ["fasta", "phylip", "nexus", "phylip-int", "nexus-int"],
00123             help = "The format of input alignment"
00124         )
00125         requiredNamed.add_argument(
00126             "-d",
00127             "--data-type",
00128             dest = "data_type",
00129             required = True,
00130             choices = ["aa", "dna"],
00131             help = "Type of data"
00132         )
00133 
```

Referenced by amas.AMAS.ParsedArgs.concat(), amas.AMAS.ParsedArgs.convert(), amas.AMAS.ParsedArgs.metapartitions(), amas.AMAS.ParsedArgs.remove(), amas.AMAS.ParsedArgs.replicate(), amas.AMAS.ParsedArgs.split(), amas.AMAS.ParsedArgs.su amas.AMAS.ParsedArgs.translate(), and amas.AMAS.ParsedArgs.trim().

Here is the caller graph for this function:


amas.AMAS.ParsedArgs.concat
amas.AMAS.ParsedArgs.convert
amas.AMAS.ParsedArgs.metapartitions
amas.AMAS.ParsedArgs.remove
amas.AMAS.ParsedArgs.replicate
amas.AMAS.ParsedArgs.add_common_args
amas.AMAS.MetaAlignment.write_metapartitions
amas.AMAS.MetaAlignment.write_out
amas.AMAS.ParsedArgs.split
amas.AMAS.ParsedArgs.summary
amas.AMAS.ParsedArgs.translate
amas.AMAS.ParsedArgs.trim

7.7.3.2 concat()

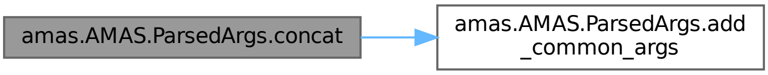
```
amas.AMAS.ParsedArgs.concat (
                self )
```

Definition at line 201 of file AMAS.py.

```
00201     def concat(self):
00202         # concat command
00203         parser = argparse.ArgumentParser(
00204             description="Concatenate input alignments"
00205         )
00206         parser.add_argument(
00207             "-p",
00208             "--concat-part",
00209             dest = "concat_part",
00210             default = "partitions.txt",
00211             help = "File name for the concatenated alignment partitions. Default: ’partitions.txt’"
00212         )
00213         parser.add_argument(
00214             "-t",
00215             "--concat-out",
00216             dest = "concat_out",
00217             default = "concatenated.out",
00218             help = "File name for the concatenated alignment. Default: 'concatenated.out'"
00219         )
00220         parser.add_argument(
00221             "-u",
00222             "--out-format",
00223             dest = "out_format",
00224             choices = ["fasta", "phylip", "nexus", "phylip-int", "nexus-int"],
00225             default = "fasta",
00226             help = "File format for the output alignment. Default: fasta"
00227         )
00228         parser.add_argument(
00229             "-y",
00230             "--part-format",
00231             dest = "part_format",
00232             choices = ["nexus", "iqtree-nexus", "raxml", "unspecified"],
00233             default = "unspecified",
00234             help = "Format of the partitions file. Default: 'unspecified'"
00235         )
00236         parser.add_argument(
00237             "-n",
00238             "--codons",
00239             dest = "codons",
00240             choices = ["none", "12", "123"],
00241             default = "none",
00242             help = "Use codon partitioning for 1st and 2nd or all three positions. Default: Don't use"
00243         )
00244         # add shared arguments
00245         self.add_common_args(parser)
00246         args = parser.parse_args(sys.argv[2:])
00247         return args
00248
```

References amas.AMAS.ParsedArgs.add_common_args().

Here is the call graph for this function:



7.7.3.3 convert()

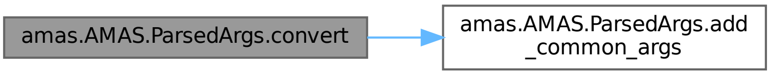
```
amas.AMAS.ParsedArgs.convert (
               self )
```

Definition at line 249 of file AMAS.py.

```
00249     def convert(self):
00250         # convert command
00251         parser = argparse.ArgumentParser(
00252             description="Convert to other file format",
00253         )
00254         parser.add_argument(
00255             "-u",
00256             "--out-format",
00257             dest = "out_format",
00258             choices = ["fasta", "phylip", "nexus", "phylip-int", "nexus-int"],
00259             default = "fasta",
00260             help = "File format for the output alignment. Default: fasta"
00261         )
00262         # add shared arguments
00263         self.add_common_args(parser)
00264         args = parser.parse_args(sys.argv[2:])
00265         return args
00266
```

References amas.AMAS.ParsedArgs.add_common_args().

Here is the call graph for this function:



7.7.3.4 get_args_dict()

```
amas.AMAS.ParsedArgs.get_args_dict (
                self )
```

Definition at line 508 of file AMAS.py.

```
00508     def get_args_dict(self):
00509         # store arguments in a dictionary
00510         command = self.args.__dict__
00511         arguments = getattr(self, self.args.command)().__dict__
00512         argument_dictionary = command.copy()
00513         argument_dictionary.update(arguments)
00514 
00515         return argument_dictionary
00516 
00517 
```

References amas.AMAS.ParsedArgs.args.

7.7.3.5 metapartitions()

```
amas.AMAS.ParsedArgs.metapartitions (
                self )
```

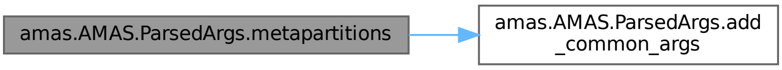
Definition at line 334 of file AMAS.py.

```
00334     def metapartitions(self):
00335         # metapartitions command
00336         parser = argparse.ArgumentParser(
00337             formatter_class=argparse.RawTextHelpFormatter,
00338             description=”’Split alignment according to a partition file, then concatenate the output.”’
00339             ”’\n\nuse case:\n”’
00340             ”’    Some utilities cannot parse partition definitions containing strides (\\) and/or discontinuous ranges.\n”’
00341             ”’    In such case, running `split` + `concat` in separate passes can convert a corresponding (super)alignment it into an\n”’
00342             ”’    equivalent compatible form with contiguous (meta)partitions; this may also require renaming metapartiton alignments\n”’
00343             ”’    and partition file entries in order to remove tags applied by each respective operation.\n\n”’
00344             ”’    `metapartitions` combines these steps into one command, with the options `--prepend` and `--no-mpan`\n”’
00345             ”’    providing additional control over the collated (meta)partition names (see their respective help entries).\n\n”’
00346             ”’    Note: in this mode, the format of the input (super)alignment file determines that of all outputs (-u|--out-format is disabled)!\n\n”’
00347         )
00348         parser.add_argument(
00349             "-p",
00350             "--concat-part",
00351             dest = "concat_part",
00352             default = "metapartitions.txt",
00353             help = "Partition file(name) for the final concatenated alignment of metapartitions. Default: ’metapartitions.txt’"
```

```
        )
        parser.add_argument(
            "-t",
            "--concat-out",
            dest = "concat_out",
            default = "concatenated-meta.out",
            help = "File name for the concatenated alignment of metapartitions. Default: 'concatenated-meta.out'"
        )
        parser.add_argument(
            "-y",
            "--part-format",
            dest = "part_format",
            choices = ["nexus", "iqtree-nexus", "raxml", "unspecified"],
            default = "unspecified",
            help = "Partitions file format for the final concatenated alignment of metapartitions. Default: 'unspecified'"
        )
        parser.add_argument(
            "-l",
            "--split-by",
            dest = "split_by",
            help = "Partition file(name) to be used for splitting the initial concatenated alignment.",
            required = True
        )
        parser.add_argument(
            "-j",
            "--remove-empty",
            dest = "remove_empty",
            action = "store_true",
            default = False,
            help = "Remove taxa with sequences composed of only undetermined characters? Default: Don't remove"
        )
        parser.add_argument(
            "--no-san",
            dest = "no_sup_aln_name",
            action = "store_true",
            default = False,
            help = ”’Don’t prepend the input (super)alignment filename to the (meta)partition-alignment filenames output by `split`”’
        )
        parser.add_argument(
            "--prepend",
            dest = "prepend_label",
            default = None,
            help = ”’Prepend <string> to the partition counter in partition file, e.g.”’
            ”’\n            --prepend <string>:  <string>_p001_metapartition_alignment_name = 1-1200 ...”’
            ”’\n                Default (None):           p001_metapartition_alignment_name = 1-1200 ...”’
            ”’\n--no-mpan + --prepend <string>:  <string>_p001 = 1-1200 ...”’
        )
        parser.add_argument(
            "--no-mpan",
            dest = "no_mpan",
            action = "store_true",
            default = False,
            help = ”’Omits (meta)partition alignment names when printing partition file, e.g.”’
            ”’\n                     --no-mpan:          p001 = 1-1200 ...”’
            ”’\n               Default (False):          p001_metapartition_alignment_name = 1-1200 ...”’
            ”’\n--prepend <string> + --no-mpan: <string>_p001 = 1-1200 ...”’
        )
        # add shared arguments
        self.add_common_args(parser)
        args = parser.parse_args(sys.argv[2:])
        return args

```

References amas.AMAS.ParsedArgs.add_common_args().

Here is the call graph for this function:



7.7.3.6 remove()

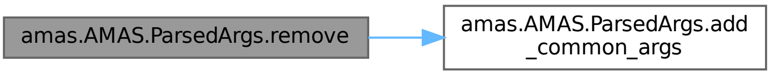
```
amas.AMAS.ParsedArgs.remove (
               self )
```

Definition at line 474 of file AMAS.py.

```
00474     def remove(self):
00475         # remove taxa command
00476         parser = argparse.ArgumentParser(
00477             description="Remove taxa from alignment",
00478         )
00479         parser.add_argument(
00480             "-x",
00481             "--taxa-to-remove",
00482             nargs = "+",
00483             type = str,
00484             dest = "taxa_to_remove",
00485             help = "Taxon/sequence names to be removed.",
00486             required = True
00487         )
00488         parser.add_argument(
00489             "-u",
00490             "--out-format",
00491             dest = "out_format",
00492             choices = ["fasta", "phylip", "nexus", "phylip-int", "nexus-int"],
00493             default = "fasta",
00494             help = "File format for the output alignment. Default: fasta"
00495         )
00496         parser.add_argument(
00497             "-g",
00498             "--out-prefix",
00499             dest = "out_prefix",
00500             default = "reduced_",
00501             help = "File name prefix for the concatenated alignment. Default: 'reduced_'"
00502         )
00503         # add shared arguments
00504         self.add_common_args(parser)
00505         args = parser.parse_args(sys.argv[2:])
00506         return args
00507
```

References amas.AMAS.ParsedArgs.add_common_args().

Here is the call graph for this function:



7.7.3.7 replicate()

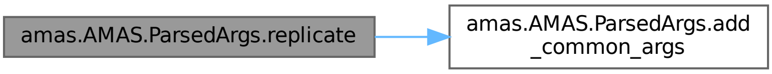
```
amas.AMAS.ParsedArgs.replicate (
               self )
```

Definition at line 267 of file AMAS.py.

```
00267     def replicate(self):
00268         # replicate command
00269         parser = argparse.ArgumentParser(
00270             description="Create replicate datasets for phylogenetic jackknife",
00271         )
00272         parser.add_argument(
00273             "-r",
00274             "--rep-aln",
00275             nargs = 2,
00276             type = int,
00277             dest = "replicate_args",
00278             help = "Create replicate data sets for phylogenetic jackknife [replicates, no alignments
      for each replicate]",
00279             required = True
00280         )
00281         parser.add_argument(
00282             "-u",
00283             "--out-format",
00284             dest = "out_format",
00285             choices = ["fasta", "phylip", "nexus", "phylip-int", "nexus-int"],
00286             default = "fasta",
00287             help = "File format for the output alignment. Default: fasta"
00288         )
00289         # add shared arguments
00290         self.add_common_args(parser)
00291         args = parser.parse_args(sys.argv[2:])
00292         return args
00293
```

References amas.AMAS.ParsedArgs.add_common_args().

Here is the call graph for this function:



7.7.3.8 split()

```
amas.AMAS.ParsedArgs.split (
                self )
```

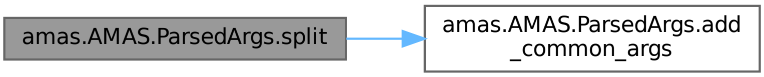
Definition at line 294 of file AMAS.py.

```
00294     def split(self):
00295         # split command
00296         parser = argparse.ArgumentParser(
00297             description="Split alignment according to a partitions file",
00298         )
00299         parser.add_argument(
00300             "-l",
00301             "--split-by",
00302             dest = "split_by",
00303             help = "File name for partitions to be used for alignment splitting.",
00304             required = True
00305         )
00306         parser.add_argument(
00307             "-j",
00308             "--remove-empty",
00309             dest = "remove_empty",
00310             action = "store_true",
00311             default = False,
00312             help = "Remove taxa with sequences composed of only undetermined characters? Default:
      Don't remove"
00313         )
00314         parser.add_argument(
00315             "-u",
00316             "--out-format",
00317             dest = "out_format",
00318             choices = ["fasta", "phylip", "nexus", "phylip-int", "nexus-int"],
00319             default = "fasta",
00320             help = "File format for the output alignment. Default: fasta"
00321         )
00322         parser.add_argument(
00323             "--no-san",
00324             dest = "no_sup_aln_name",
00325             action = "store_true",
00326             default = False,
00327             help = ”’Don’t prepend the input (super)alignment filename to the partition-alignment
      filenames output by `split`”’
00328         )
00329         # add shared arguments
00330         self.add_common_args(parser)
00331         args = parser.parse_args(sys.argv[2:])
00332         return args
00333
```

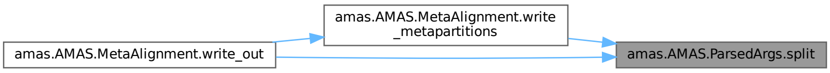
References amas.AMAS.ParsedArgs.add_common_args().

Referenced by amas.AMAS.MetaAlignment.write_metapartitions(), and amas.AMAS.MetaAlignment.write_out().

Here is the call graph for this function:


amas.AMAS.ParsedArgs.split
amas.AMAS.ParsedArgs.add
_common_args

Here is the caller graph for this function:


amas.AMAS.MetaAlignment.write_out
amas.AMAS.MetaAlignment.write
_metapartitions
amas.AMAS.ParsedArgs.split

7.7.3.9 summary()

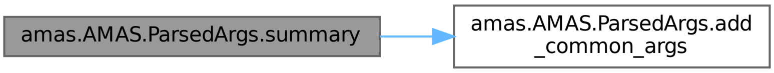
```
amas.AMAS.ParsedArgs.summary (
                self )
```

Definition at line 176 of file AMAS.py.

```
00176     def summary(self):
00177        # summary command
00178         parser = argparse.ArgumentParser(
00179             description="Write alignment summary",
00180         )
00181         parser.add_argument(
00182             "-o",
00183             "--summary-out",
00184             dest = "summary_out",
00185             default = "summary.txt",
00186             help = "File name for the alignment summary. Default: ’summary.txt’"
00187         )
00188         parser.add_argument(
00189             "-s",
00190             "--by-taxon",
00191             dest = "by_taxon_summary",
00192             action = "store_true",
00193             default = False,
00194             help = "In addition to alignment summary, write by sequence/taxon summaries. Default:
      Don’t write"
00195         )
00196         # add shared arguments
00197         self.add_common_args(parser)
00198         args = parser.parse_args(sys.argv[2:])
00199         return args
00200
```

References amas.AMAS.ParsedArgs.add_common_args().

Here is the call graph for this function:


amas.AMAS.ParsedArgs.summary
amas.AMAS.ParsedArgs.add_common_args

7.7.3.10 translate()

```
amas.AMAS.ParsedArgs.translate (
                self )
```

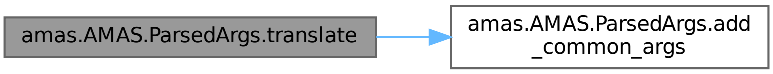
Definition at line 416 of file AMAS.py.

```
00416     def translate(self):
00417         # translate command
00418         parser = argparse.ArgumentParser(
00419             formatter_class=argparse.RawTextHelpFormatter,
00420             description="Translate a protein-coding DNA alignment into amino acids"
00421         )
00422         parser.add_argument(
00423             "-b",
00424             "--code",
00425             type = int,
00426             dest = "genetic_code",
00427             choices = [1, 2, 3, 4, 5, 6, 9, 10, 11, 12, 13, 14, 16, 21, 22, 23, 24, 25, 26],
00428             default = 1,
00429             help = ”’\nNCBI genetic code to use (Default: 1):”’
00430 ”’
00431    1. The Standard Code
00432    2. The Vertebrate Mitochondrial Code
00433    3. The Yeast Mitochondrial Code
00434    4. The Mold, Protozoan, and Coelenterate Mitochondrial Code and the Mycoplasma/Spiroplasma Code
00435    5. The Invertebrate Mitochondrial Code
00436    6. The Ciliate, Dasycladacean and Hexamita Nuclear Code
00437    9. The Echinoderm and Flatworm Mitochondrial Code
00438   10. The Euplotid Nuclear Code
00439   11. The Bacterial, Archaeal and Plant Plastid Code
00440   12. The Alternative Yeast Nuclear Code
00441   13. The Ascidian Mitochondrial Code
00442   14. The Alternative Flatworm Mitochondrial Code
00443   16. Chlorophycean Mitochondrial Code
00444   21. Trematode Mitochondrial Code
00445   22. Scenedesmus obliquus Mitochondrial Code
00446   23. Thraustochytrium Mitochondrial Code
00447   24. Pterobranchia Mitochondrial Code
00448   25. Candidate Division SR1 and Gracilibacteria Code
00449   26. Pachysolen tannophilus Nuclear Code\n
00450 ”’
00451         )
00452         parser.add_argument(
00453             "-k",
00454             "--reading-frame",
00455             type = int,
00456             dest = "reading_frame",
00457             choices = [1, 2, 3],
00458             default = 1,
00459             help = "Number specifying reading frame; i.e. ’2’ means codons start at the second
      character of the alignment. Default: 1",
00460         )
00461         parser.add_argument(
00462             "-u",
00463             "--out-format",
00464             dest = "out_format",
00465             choices = ["fasta", "phylip", "nexus", "phylip-int", "nexus-int"],
00466             default = "fasta",
00467             help = "File format for the output alignment. Default: fasta"
00468         )
00469         # add shared arguments
```

```
        self.add_common_args(parser)
        args = parser.parse_args(sys.argv[2:])
        return args

```

References amas.AMAS.ParsedArgs.add_common_args().

Here is the call graph for this function:


amas.AMAS.ParsedArgs.translate
amas.AMAS.ParsedArgs.add_common_args

7.7.3.11 trim()

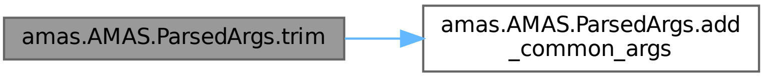
```
amas.AMAS.ParsedArgs.trim (
              self )
```

Definition at line 134 of file AMAS.py.

```
00134     def trim(self):
00135         # trim command
00136         parser = argparse.ArgumentParser(
00137             formatter_class=argparse.RawDescriptionHelpFormatter,
00138             description=”’Trim alignment by occupancy. Optionally removes sites that are not parsimony informative.”’
00139             ”’\nCAUTION: when running on amino acids stop codons marked with * will be treated as missing data!”’
00140         )
00141         parser.add_argument(
00142             "-u",
00143             "--out-format",
00144             dest = "out_format",
00145             choices = ["fasta", "phylip", "nexus", "phylip-int", "nexus-int"],
00146             default = "fasta",
00147             help = "File format for the output alignment. Default: fasta"
00148         )
00149         parser.add_argument(
00150             "-o",
00151             "--trim-out",
00152             dest = "trim_out",
00153             help = "File name for the trimmed alignment when providing a single file as input."
00154         )
00155         parser.add_argument(
00156             "-t",
00157             "--trim-fraction",
00158             type = proportion,
00159             dest = "trim_fraction",
00160             default = 0.6,
00161             help = "Columns in the alignments with occupancy lower than this value will be removed. Default: 0.6"
00162         )
00163         parser.add_argument(
00164             "-p",
00165             "--retain-only-parsimony-sites",
00166             dest = "parsimony_check",
00167             action = "store_true",
00168             default = False,
00169             help = "Only write parsimony informative columns in trimmed alignment Default: write all columns"
00170         )
00171         # add shared arguments
00172         self.add_common_args(parser)
00173         args = parser.parse_args(sys.argv[2:])
00174         return args
00175 
```

References amas.AMAS.ParsedArgs.add_common_args().

Here is the call graph for this function:


amas.AMAS.ParsedArgs.trim
amas.AMAS.ParsedArgs.add_common_args

7.7.4 Member Data Documentation

7.7.4.1 args

`amas.AMAS.ParsedArgs.args`

Definition at line 79 of file AMAS.py.

Referenced by amas.AMAS.ParsedArgs.get_args_dict().

The documentation for this class was generated from the following file:

- amas/AMAS.py

Chapter 8

File Documentation

8.1 amas/__init__.py File Reference

Namespaces

- namespace amas

Variables

- str amas.__author__ = 'Marek Borowiec'
- str amas.__email__ = 'petiolus@gmail.com'
- str amas.__version__ = '1.02'
- amas.__all__ = dir()

8.2 __init__.py

Go to the documentation of this file.

```
# -*- coding: utf-8 -*-

__author__ = ’Marek Borowiec’
__email__ = ’petiolus@gmail.com’
__version__ = ’1.02’
__all__ = dir()
```

8.3 amas/AMAS.py File Reference

Classes

- class amas.AMAS.ParsedArgs
- class amas.AMAS.FileHandler
- class amas.AMAS.FileParser
- class amas.AMAS.Alignment
- class amas.AMAS.AminoAcidAlignment
- class amas.AMAS.DNAAlignment
- class amas.AMAS.MetaAlignment

Namespaces

- namespace amas
- namespace amas.AMAS

Functions

- amas.AMAS.proportion (x)
- amas.AMAS.main ()
- amas.AMAS.run ()

8.4 AMAS.py

Go to the documentation of this file.

```
#! /usr/bin/env python3
# -*- coding: utf-8 -*-
# vim:fileencoding=utf-8

#   Program to calculate various statistics on a multiple sequence alignment
#   and allow efficient manipulation of phylogenomic data sets

#   Copyright (C) 2015 Marek Borowiec

#   This program is free software: you can redistribute it and/or modify
#   it under the terms of the GNU General Public License as published by
#   the Free Software Foundation, either version 3 of the License, or
#   (at your option) any later version.

#   This program is distributed in the hope that it will be useful,
#   but WITHOUT ANY WARRANTY; without even the implied warranty of
#   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#   GNU General Public License for more details.

#   You should have received a copy of the GNU General Public License
#   along with this program.  If not, see <http://www.gnu.org/licenses/>.

"""
This stand-alone program allows manipulations of multiple sequence
alignments. It supports sequential FASTA, PHYLIP, NEXUS, and interleaved PHYLIP
and NEXUS formats for DNA and aino acid sequences. It can print summary statistics,
convert among formats, and concatenate alignments.

Current statistics include the number of taxa, alignment length, total number
of matrix cells, overall number of undetermined characters, percent of missing
data, AT and GC contents (for DNA alignments), number and proportion of
variable sites, number and proportion of parsimony informative sites,
and counts of all characters present in the relevant (nucleotide or amino acid) alphabet.
"""


import argparse, multiprocessing as mp, re, sys
from random import sample
from os import path, remove
from collections import defaultdict, Counter
from itertools import compress

def proportion(x):
    # needed to prevent input of invalid floats in trim mode
    x = float(x)
    if x < 0.0 or x > 1.0:
        raise argparse.ArgumentTypeError("%r not in range [0.0, 1.0]" % (x,))
    return x

class ParsedArgs:

    def __init__(self):
        parser = argparse.ArgumentParser(
            usage=”’AMAS <command> [<args>]

The AMAS commands are:
  concat              Concatenate input alignments.
  convert             Convert to other file format.
  replicate           Create replicate data sets for phylogenetic jackknife.
  split               Split alignment according to a partitions file.
  summary             Write alignment summary.
  remove            Remove taxa from alignment.
  translate         Translate DNA alignment into protein alignment.
  trim              Remove columns from alignment.
  metapartitions    Runs `split` and concatenates the output.


Use AMAS <command> -h for help with arguments of the command of interest
”’
        )

        parser.add_argument(
            "command",
            help="Subcommand to run"
        )

        # parse_args defaults to [1:] for args, but you need to
        # exclude the rest of the args too, or validation will fail
        self.args = parser.parse_args(sys.argv[1:2])
        if not hasattr(self, self.args.command):
            print(’Unrecognized command’)
            parser.print_help()
            exit(1)
        # use dispatch pattern to invoke method with same name
        getattr(self, self.args.command)()

    def add_common_args(self, parser):
        # define required arguments for every command
        requiredNamed = parser.add_argument_group(’required arguments’)
        parser.add_argument(
            "-e",
            "--check-align",
            dest = "check_align",
            action = "store_true",
            default = False,
            help = "Check if input sequences are aligned. Default: no check"
        )
        parser.add_argument(
            # parallelization is used for file parsing and calculating summary stats
            "-c",
            "--cores",
            dest = "cores",
            default = 1,
            help = "Number of cores used. Default: 1"
        )

        requiredNamed.add_argument(
            "-i",
            "--in-files",
            nargs = "+",
            type = str,
            dest = "in_files",
            required = True,
            help = """Alignment files to be taken as input.
            You can specify multiple files using wildcards (e.g. --in-files *fasta)"""
        )
        requiredNamed.add_argument(
            "-f",
            "--in-format",
            dest = "in_format",
            required = True,
            choices = ["fasta", "phylip", "nexus", "phylip-int", "nexus-int"],
            help = "The format of input alignment"
        )
        requiredNamed.add_argument(
            "-d",
            "--data-type",
            dest = "data_type",
            required = True,
            choices = ["aa", "dna"],
            help = "Type of data"
        )

    def trim(self):
        # trim command
        parser = argparse.ArgumentParser(
            formatter_class=argparse.RawDescriptionHelpFormatter,
            description=”’Trim alignment by occupancy. Optionally removes sites that are not parsimony informative.”’
            ”’\nCAUTION: when running on amino acids stop codons marked with * will be treated as missing data!”’
        )
        parser.add_argument(
            "-u",
            "--out-format",
            dest = "out_format",
            choices = ["fasta", "phylip", "nexus", "phylip-int", "nexus-int"],
            default = "fasta",
            help = "File format for the output alignment. Default: fasta"
        )
        parser.add_argument(
            "-o",
            "--trim-out",
            dest = "trim_out",
            help = "File name for the trimmed alignment when providing a single file as input."
        )
        parser.add_argument(
            "-t",
            "--trim-fraction",
            type = proportion,
            dest = "trim_fraction",
            default = 0.6,
            help = "Columns in the alignments with occupancy lower than this value will be removed. Default: 0.6"
        )
        parser.add_argument(
            "-p",
            "--retain-only-parsimony-sites",
            dest = "parsimony_check",
            action = "store_true",
            default = False,
            help = "Only write parsimony informative columns in trimmed alignment Default: write all columns"
        )
        # add shared arguments
        self.add_common_args(parser)
        args = parser.parse_args(sys.argv[2:])
        return args

    def summary(self):
       # summary command
        parser = argparse.ArgumentParser(
            description="Write alignment summary",
        )
        parser.add_argument(
            "-o",
            "--summary-out",
            dest = "summary_out",
            default = "summary.txt",
            help = "File name for the alignment summary. Default: 'summary.txt'"
        )
        parser.add_argument(
            "-s",
            "--by-taxon",
            dest = "by_taxon_summary",
            action = "store_true",
            default = False,
            help = "In addition to alignment summary, write by sequence/taxon summaries. Default: Don't write"
        )
        # add shared arguments
        self.add_common_args(parser)
        args = parser.parse_args(sys.argv[2:])
        return args

    def concat(self):
        # concat command
        parser = argparse.ArgumentParser(
            description="Concatenate input alignments"
        )
        parser.add_argument(
            "-p",
            "--concat-part",
            dest = "concat_part",
            default = "partitions.txt",
            help = "File name for the concatenated alignment partitions. Default: 'partitions.txt'"
        )
        parser.add_argument(
            "-t",
            "--concat-out",
            dest = "concat_out",
            default = "concatenated.out",
            help = "File name for the concatenated alignment. Default: 'concatenated.out'"
        )
        parser.add_argument(
            "-u",
            "--out-format",
            dest = "out_format",
            choices = ["fasta", "phylip", "nexus", "phylip-int", "nexus-int"],
            default = "fasta",
            help = "File format for the output alignment. Default: fasta"
        )
        parser.add_argument(
            "-y",
            "--part-format",
            dest = "part_format",
            choices = ["nexus", "iqtree-nexus", "raxml", "unspecified"],
            default = "unspecified",
            help = "Format of the partitions file. Default: 'unspecified'"
        )
        parser.add_argument(
            "-n",
            "--codons",
            dest = "codons",
            choices = ["none", "12", "123"],
            default = "none",
            help = "Use codon partitioning for 1st and 2nd or all three positions. Default: Don't use"
        )
        # add shared arguments
        self.add_common_args(parser)
        args = parser.parse_args(sys.argv[2:])
        return args

    def convert(self):
        # convert command
        parser = argparse.ArgumentParser(
            description="Convert to other file format",
        )
        parser.add_argument(
            "-u",
            "--out-format",
            dest = "out_format",
            choices = ["fasta", "phylip", "nexus", "phylip-int", "nexus-int"],
            default = "fasta",
            help = "File format for the output alignment. Default: fasta"
        )
        # add shared arguments
        self.add_common_args(parser)
        args = parser.parse_args(sys.argv[2:])
        return args

    def replicate(self):
        # replicate command
        parser = argparse.ArgumentParser(
            description="Create replicate datasets for phylogenetic jackknife",
        )
        parser.add_argument(
            "-r",
            "--rep-aln",
            nargs = 2,
            type = int,
            dest = "replicate_args",
            help = "Create replicate data sets for phylogenetic jackknife [replicates, no alignments for each replicate]",
            required = True
        )
        parser.add_argument(
            "-u",
            "--out-format",
            dest = "out_format",
            choices = ["fasta", "phylip", "nexus", "phylip-int", "nexus-int"],
            default = "fasta",
            help = "File format for the output alignment. Default: fasta"
        )
        # add shared arguments
        self.add_common_args(parser)
        args = parser.parse_args(sys.argv[2:])
        return args

    def split(self):
        # split command
        parser = argparse.ArgumentParser(
            description="Split alignment according to a partitions file",
        )
        parser.add_argument(
            "-l",
            "--split-by",
            dest = "split_by",
            help = "File name for partitions to be used for alignment splitting.",
            required = True
        )
        parser.add_argument(
            "-j",
            "--remove-empty",
            dest = "remove_empty",
            action = "store_true",
            default = False,
            help = "Remove taxa with sequences composed of only undetermined characters? Default: Don't remove"
        )
        parser.add_argument(
            "-u",
            "--out-format",
            dest = "out_format",
            choices = ["fasta", "phylip", "nexus", "phylip-int", "nexus-int"],
            default = "fasta",
            help = "File format for the output alignment. Default: fasta"
        )
        parser.add_argument(
            "--no-san",
            dest = "no_sup_aln_name",
            action = "store_true",
            default = False,
            help = ”’Don’t prepend the input (super)alignment filename to the partition-alignment filenames output by `split`”’
        )
        # add shared arguments
        self.add_common_args(parser)
        args = parser.parse_args(sys.argv[2:])
        return args

    def metapartitions(self):
        # metapartitions command
        parser = argparse.ArgumentParser(
            formatter_class=argparse.RawTextHelpFormatter,
            description=”’Split alignment according to a partition file, then concatenate the output.”’
            ”’\n\nuse case:\n”’
            ”’    Some utilities cannot parse partition definitions containing strides (\\) and/or discontinuous ranges.\n”’
            ”’    In such case, running `split` + `concat` in separate passes can convert a corresponding (super)alignment it into an\n”’
            ”’    equivalent compatible form with contiguous (meta)partitions; this may also require renaming metapartiton alignments\n”’
            ”’    and partition file entries in order to remove tags applied by each respective operation.\n\n”’
            ”’    `metapartitions` combines these steps into one command, with the options `--prepend` and `--no-mpan`\n”’
            ”’    providing additional control over the collated (meta)partition names (see their respective help entries).\n\n”’
            ”’    Note: in this mode, the format of the input (super)alignment file determines that of all outputs (-u|--out-format is disabled)!\n\n”’
        )
        parser.add_argument(
            "-p",
            "--concat-part",
            dest = "concat_part",
            default = "metapartitions.txt",
            help = "Partition file(name) for the final concatenated alignment of metapartitions. Default: ’metapartitions.txt’"
        )
        parser.add_argument(
            "-t",
            "--concat-out",
            dest = "concat_out",
            default = "concatenated-meta.out",
            help = "File name for the concatenated alignment of metapartitions. Default: ’concatenated-meta.out’"
        )
        parser.add_argument(
            "-y",
            "--part-format",
            dest = "part_format",
            choices = ["nexus", "iqtree-nexus", "raxml", "unspecified"],
            default = "unspecified",
            help = "Partitions file format for the final concatenated alignment of metapartitions. Default: ’unspecified’"
        )
        parser.add_argument(
            "-l",
            "--split-by",
            dest = "split_by",
            help = "Partition file(name) to be used for splitting the initial concatenated alignment.",
            required = True
        )
        parser.add_argument(
            "-j",
            "--remove-empty",
            dest = "remove_empty",
            action = "store_true",
            default = False,
            help = "Remove taxa with sequences composed of only undetermined characters? Default: Don’t remove"
        )
        parser.add_argument(
            "--no-san",
            dest = "no_sup_aln_name",
            action = "store_true",
            default = False,
            help = ”’Don’t prepend the input (super)alignment filename to the (meta)partition-alignment filenames output by `split`”’
        )
        parser.add_argument(
            "--prepend",
            dest = "prepend_label",
            default = None,
            help = ”’Prepend <string> to the partition counter in partition file, e.g.”’
            ”’\n            --prepend <string>:  <string>_p001_metapartition_alignment_name = 1-1200 ...”’
            ”’\n                Default (None):           p001_metapartition_alignment_name = 1-1200 ...”’
            ”’\n--no-mpan + --prepend <string>:  <string>_p001 = 1-1200 ...”’
        )
        parser.add_argument(
            "--no-mpan",
            dest = "no_mpan",
            action = "store_true",
            default = False,
            help = ”’Omits (meta)partition alignment names when printing partition file, e.g.”’
            ”’\n                     --no-mpan:          p001 = 1-1200 ...”’
            ”’\n               Default (False):          p001_metapartition_alignment_name = 1-1200 ...”’
            ”’\n--prepend <string> + --no-mpan: <string>_p001 = 1-1200 ...”’
        )
        # add shared arguments
        self.add_common_args(parser)
        args = parser.parse_args(sys.argv[2:])
        return args

    def translate(self):
        # translate command
        parser = argparse.ArgumentParser(
            formatter_class=argparse.RawTextHelpFormatter,
            description="Translate a protein-coding DNA alignment into amino acids"
        )
        parser.add_argument(
            "-b",
            "--code",
            type = int,
            dest = "genetic_code",
            choices = [1, 2, 3, 4, 5, 6, 9, 10, 11, 12, 13, 14, 16, 21, 22, 23, 24, 25, 26],
            default = 1,
            help = ”’\nNCBI genetic code to use (Default: 1):”’
”’
   1. The Standard Code
   2. The Vertebrate Mitochondrial Code
   3. The Yeast Mitochondrial Code
   4. The Mold, Protozoan, and Coelenterate Mitochondrial Code and the Mycoplasma/Spiroplasma Code
   5. The Invertebrate Mitochondrial Code
   6. The Ciliate, Dasycladacean and Hexamita Nuclear Code
   9. The Echinoderm and Flatworm Mitochondrial Code
  10. The Euplotid Nuclear Code
  11. The Bacterial, Archaeal and Plant Plastid Code
  12. The Alternative Yeast Nuclear Code
  13. The Ascidian Mitochondrial Code
  14. The Alternative Flatworm Mitochondrial Code
  16. Chlorophycean Mitochondrial Code
  21. Trematode Mitochondrial Code
  22. Scenedesmus obliquus Mitochondrial Code
  23. Thraustochytrium Mitochondrial Code
  24. Pterobranchia Mitochondrial Code
  25. Candidate Division SR1 and Gracilibacteria Code
  26. Pachysolen tannophilus Nuclear Code\n
”’
        )
        parser.add_argument(
            "-k",
            "--reading-frame",
            type = int,
            dest = "reading_frame",
            choices = [1, 2, 3],
            default = 1,
            help = "Number specifying reading frame; i.e. ’2’ means codons start at the second character of the alignment. Default: 1",
        )
        parser.add_argument(
            "-u",
            "--out-format",
            dest = "out_format",
            choices = ["fasta", "phylip", "nexus", "phylip-int", "nexus-int"],
            default = "fasta",
            help = "File format for the output alignment. Default: fasta"
        )
        # add shared arguments
        self.add_common_args(parser)
        args = parser.parse_args(sys.argv[2:])
        return args

    def remove(self):
        # remove taxa command
        parser = argparse.ArgumentParser(
            description="Remove taxa from alignment",
        )
        parser.add_argument(
            "-x",
            "--taxa-to-remove",
            nargs = "+",
            type = str,
            dest = "taxa_to_remove",
            help = "Taxon/sequence names to be removed.",
            required = True
        )
        parser.add_argument(
            "-u",
            "--out-format",
            dest = "out_format",
            choices = ["fasta", "phylip", "nexus", "phylip-int", "nexus-int"],
            default = "fasta",
            help = "File format for the output alignment. Default: fasta"
        )
        parser.add_argument(
            "-g",
            "--out-prefix",
            dest = "out_prefix",
            default = "reduced_",
            help = "File name prefix for the concatenated alignment. Default: 'reduced_'"
        )
        # add shared arguments
        self.add_common_args(parser)
        args = parser.parse_args(sys.argv[2:])
        return args

    def get_args_dict(self):
        # store arguments in a dictionary
        command = self.args.__dict__
        arguments = getattr(self, self.args.command)().__dict__
        argument_dictionary = command.copy()
        argument_dictionary.update(arguments)

        return argument_dictionary


class FileHandler:
    """Define file handle that closes when out of scope"""

    def __init__(self, file_name):
        self.file_name = file_name

    def __enter__(self):
        try:
            self.in_file = open(self.file_name, "r", encoding="utf-8")
        except FileNotFoundError:
            print("ERROR: File '" + self.file_name + "' not found.")
            sys.exit()
        return self.in_file

    def __exit__(self, *args):
        self.in_file.close()

    def get_file_name(self):
        return self.file_name

class FileParser:
    """Parse file contents and return sequences and sequence names"""

    def __init__(self, in_file):
        self.in_file = in_file
        with FileHandler(in_file) as handle:
            self.in_file_lines = handle.read().rstrip("\r\n")

    def fasta_parse(self):
        # use regex to parse names and sequences in sequential fasta files
        matches = re.finditer(
            r"^>(.*[^$])([^>]*)",
            self.in_file_lines, re.MULTILINE
        )
        records = {}

        for match in matches:
            name_match = match.group(1).replace("\n", "")
            seq_match = match.group(2).replace("\n", "").upper()
            seq_match = self.translate_ambiguous(seq_match)
```

```
                records[name_match] = seq_match

        return records

    def phylip_parse(self):
        # use regex to parse names and sequences in sequential phylip files
        matches = re.finditer(
            r"^(\s+)?(\S+)\s+([A-Za-z*?.{}-]+)",
            self.in_file_lines, re.MULTILINE
        )

        records = {}

        for match in matches:
            name_match = match.group(2).replace("\n", "")
            seq_match = match.group(3).replace("\n", "").upper()
            seq_match = self.translate_ambiguous(seq_match)
            records[name_match] = seq_match

        return records

    def phylip_interleaved_parse(self):
    # use regex to parse names and sequences in interleaved phylip files
        tax_chars_matches = re.finditer(
            r"^(\s+)?([0-9]+)[ \t]+([0-9]+)",
            self.in_file_lines, re.MULTILINE
        )
        name_matches = re.finditer(
            r"^(\s+)?(\S+)[ \t]+[A-Za-z*?.{}-]+",
            self.in_file_lines, re.MULTILINE
        )
        seq_matches = re.finditer(
            r"(^(\s+)?\S+[ \t]+|^)([A-Za-z*?.{}-]+)$",
            self.in_file_lines, re.MULTILINE
        )
        # get number of taxa and chars
        for match in tax_chars_matches:
            tax_match = match.group(2)
            chars_match = match.group(3)

        # initiate lists for taxa names and sequence strings on separate lines
        taxa = []
        sequences = []
        # initiate a dictionary for the name:sequence records
        records = {}
        # initiate a counter to keep track of sequences strung together
        # from separate lines
        counter = 0

        for match in name_matches:
            name_match = match.group(2).replace("\n", "")
            taxa.append(name_match)

        for match in seq_matches:
            seq_match = match.group(3).replace("\n", "").upper()
            seq_match = self.translate_ambiguous(seq_match)
            sequences.append(seq_match)
        # try parsing PHYLUCE-style interleaved phylip
        if len(taxa) != int(tax_match):
            taxa = []
            sequences = []
            matches = re.finditer(
                r"(^(\s+)?(\S+)( ){2,}|^\s+)([ A-Za-z*?.{}-]+)",
                self.in_file_lines, re.MULTILINE
            )

            for match in matches:
                try:
                    name_match = match.group(3).replace("\n", "")
                    taxa.append(name_match)
                except AttributeError:
                    pass
                seq_match = match.group(5).replace("\n", "").upper()
                seq_match = "".join(seq_match.split())
                seq_match = self.translate_ambiguous(seq_match)
                sequences.append(seq_match)

        for taxon_no in range(len(taxa)):
            sequence = ""
            for index in range(counter,len(sequences),len(taxa)):
                sequence += sequences[index]

            records[taxa[taxon_no]] = sequence
            counter += 1

        return records

    def nexus_parse(self):
        # use regex to parse names and sequences in sequential nexus files
        # find the matrix block
        matches = re.finditer(
            r"(\s+)?(MATRIX\n|matrix\n|MATRIX\r\n|matrix\r\n)(.*?;)",
            self.in_file_lines, re.DOTALL
        )

        records = {}
        # get names and sequences from the matrix block

        for match in matches:
            matrix_match = match.group(3)
            seq_matches = re.finditer(
                r"^(\s+)?[’]?(\S+\s\S+|\S+)[’]?\s+([A-Za-z*?.{}-]+)($|\s+\[[0-9]+\]$)",
                matrix_match, re.MULTILINE
            )

            for match in seq_matches:
                name_match = match.group(2).replace("\n", "")
                seq_match = match.group(3).replace("\n", "").upper()
                seq_match = self.translate_ambiguous(seq_match)
                records[name_match] = seq_match

        return records

    def nexus_interleaved_parse(self):
        # use regex to parse names and sequences in sequential nexus files
        # find the matrix block
        matches = re.finditer(
            r"(\s+)?(MATRIX\n|matrix\n|MATRIX\r\n|matrix\r\n)(.*?;)",
            self.in_file_lines, re.DOTALL
        )
        # initiate lists for taxa names and sequence strings on separate lines
        taxa = []
        sequences = []
        # initiate a dictionary for the name:sequence records
        records = {}

        for match in matches:
            matrix_match = match.group(3)
            # get names and sequences from the matrix block
            seq_matches = re.finditer(
                r"^(\s+)?[’]?(\S+\s\S+|\S+)[’]?\s+([A-Za-z*?.{}-]+)($|\s+\[[0-9]+\]$)",
                matrix_match, re.MULTILINE
            )

            for match in seq_matches:
                name_match = match.group(2)
                if name_match not in taxa:
                    taxa.append(name_match)
                seq_match = match.group(3)

                sequences.append(seq_match)

        # initiate a counter to keep track of sequences strung together
        # from separate lines
        counter = 0

        for taxon_no in range(len(taxa)):

            full_length_sequence = "".join([sequences[index] for index in range(counter,len(sequences),len(taxa))])
            records[taxa[taxon_no]] = self.translate_ambiguous(full_length_sequence).replace("\n", "").upper()
            counter += 1

        return records

    def translate_ambiguous(self, seq):
        # translate ambiguous characters from curly bracket format
        # to single letter format
        # also remove spaces from sequences
        seq = seq.replace("{GT}", "K")
        seq = seq.replace("{AC}", "M")
        seq = seq.replace("{AG}", "R")
        seq = seq.replace("{CT}", "Y")
        seq = seq.replace("{CG}", "S")
        seq = seq.replace("{AT}", "W")
        seq = seq.replace("{CGT}", "B")
        seq = seq.replace("{ACG}", "V")
        seq = seq.replace("{ACT}", "H")
        seq = seq.replace("{AGT}", "D")
        seq = seq.replace("{GATC}", "N")
        seq = seq.replace(" ", "")

        return seq

    def partitions_parse(self):
        # parse partitions file using regex
        # original: `matches = re.finditer(r"^(\s+)?([^ =]+)[ =]+([\\0-9, -]+)", self.in_file_lines, re.MULTILINE)`
        # new version: more permissive -> handles PartionFinder/RAxML/(IQ-TREE 2)best_scheme.nex format partition files
        matches = re.finditer(
            r"""^[ \t]*                                # start of line w/ zero-or-more (just) whitespaces/tabs
                (
                 (?P<nexus>charset[ ]+)                # case 1: (IQ-TREE 2)best_scheme.nex partition directive; partition name
                 |
                 (?P<raxml>[A-Za-z0-9_+\.]+,[ \t]+)    # case 2: RAxML/RAxML-NG model(+other pars); partition name
                )?
                (?P<partition_name>[A-Za-z0-9_@*&()\[\]{}'"%#|-]+) # partition name (handles various metcharacters, excludes [\s],<>?/:;~`\^$!)
                [ ]*=[ ]*                              # whitespace-padded (or unpadded) '=': (IQ-TREE 2)best_scheme.nex compatabiliy
                (?P<numbers>[\\0-9, -]+)               # position ranges w/ stride (multiple intervals; from original regex)
                (?P<nexus_term>[ ]*[;])?               # whitespace-prepended (or unprepended) ';' (nexus terminator)
            """,
            self.in_file_lines,
            re.MULTILINE | re.VERBOSE
        )

        # initiate list to store dictionaries with lists
        # of slice positions as values
        partitions = []
        add_to_partitions = partitions.append

        for match in matches:
            # initiate dictionary of partition name as key
            dict_of_dicts = {}
            # and list of dictionaries with slice positions
            list_of_dicts = []
            add_to_list_of_dicts = list_of_dicts.append
            # get parition name and numbers from parsed partition strings
            partition_name = match.group('partition_name')
            numbers = match.group('numbers')
            # remove any whitespace padding '-' (to be consistent with partition-writing format)
            numbers = re.sub(r"[ ]*-[ ]*", "-", numbers)
            # find all numbers that will be used to parse positions
            positions = re.findall(r"([^ ,]+)", numbers)

            for position in positions:
                # create dictionary for slicing input sequence
                # conditioning on whether positions are represented
                # by range, range with stride, or single number
                pos_dict = {}

                if "-" in position:
                    m = re.search(r"([0-9]+)-([0-9]+)", position)
                    pos_dict["start"] = int(m.group(1)) - 1
                    pos_dict["stop"] = int(m.group(2))
                else:
                    pos_dict["start"] = int(position) - 1
                    pos_dict["stop"] = int(position)

                if "\\" in position:
                    # Note: the value of `N` in `...\N` isn't read: the script simply assumes `N` is consistent with the number of
                    # increments per interval when the alignment is parsed with a stride of 3 (designating each cpos).
                    # E.g. For the partition file:
                    #       ...`1-N\2`
                    #       ...`2-N\2`
                    #       ...`(N+1)-M\2`
                    #       ...`(N+2)-M\2`
                    # 3'cpos are ignored due to the absence of intervals `3-N...', `(N+3)-M...', not because the associated stride values are`\2`
                    pos_dict["stride"] = 3
                elif "\\" not in position:
                    pos_dict["stride"] = 1

                add_to_list_of_dicts(pos_dict)

            dict_of_dicts[partition_name] = list_of_dicts
            add_to_partitions(dict_of_dicts)

        return partitions


```

```
class Alignment:
    """Base class: Gets in parsed sequences as input and summarizes their stats.
    Based on the data type, the subclasses AminoAcidAlignment & DNAAlignment define the attributes:
    `alphabet`, `missing_ambiguous_chars`, `missing_chars`, `non_alphabet`
    """

    def __init__(self, in_file, in_format, data_type):
        # initialize alignment class with parsed records and alignment name as arguments,
        # create empty lists for list of sequences, sites without
        # ambiguous or missing characters, and initialize variable for the number
        # of parsimony informative sites
        self.in_file = in_file
        self.in_format = in_format
        self.data_type = data_type

        self.parsed_aln = self.get_parsed_aln()

    def __str__(self):
        #  purpose of override? (originally returned method object)
        return self.get_name()

    def get_aln_input(self):
        # open and parse input file
        aln_input = FileParser(self.in_file)
        return aln_input

    def get_parsed_aln(self):
        # parse according to the given format
        aln_input = self.get_aln_input()
        if self.in_format == "fasta":
            parsed_aln = aln_input.fasta_parse()
        elif self.in_format == "phylip":
            parsed_aln = aln_input.phylip_parse()
        elif self.in_format == "phylip-int":
            parsed_aln = aln_input.phylip_interleaved_parse()
        elif self.in_format == "nexus":
            parsed_aln = aln_input.nexus_parse()
        elif self.in_format == "nexus-int":
            parsed_aln = aln_input.nexus_interleaved_parse()

        return parsed_aln

    def summarize_alignment(self):
        # call methods to create sequences list, matrix, sites without ambiguous or
        # missing characters; get and summarize alignment statistics
        summary = []
        self.length = str(self.get_alignment_length())
        self.matrix = self.matrix_creator()
        self.no_missing_ambiguous = self.get_sites_no_missing_ambiguous()
        self.variable_sites = self.get_variable()
        self.prop_variable = self.get_prop_variable()
        self.parsimony_informative = self.get_parsimony_informative()
        self.prop_parsimony = self.get_prop_parsimony()
        self.missing_records = self.get_missing_from_parsed()
        name = str(self.get_name())
        taxa_no = str(self.get_taxa_no())
        cells = str(self.get_matrix_cells())
        missing = str(self.get_missing())
        missing_percent = str(self.get_missing_percent())
        self.check_data_type()
        summary = [
            name,
            taxa_no,
            self.length,
            cells,
            missing,
            missing_percent,
            str(self.variable_sites),
            str(self.prop_variable),
            str(self.parsimony_informative),
            str(self.prop_parsimony)
        ]
        return summary

    def summarize_alignment_by_taxa(self):
        # get summary for all taxa/sequences in alignment
        per_taxon_summary = []
        taxa_no = self.get_taxa_no()
        self.missing_records = self.get_missing_from_parsed()
        self.length = self.get_alignment_length()
        lengths = (self.length for i in range(taxa_no))
        name = self.get_name()
        names = (name for i in range(taxa_no))
        taxa_names = (
            taxon.replace(" ", "_").replace(".", "_").replace("’", "")
            for taxon, missing_count, missing_percent in self.missing_records
        )
        missing = (missing_count for taxon, missing_count, missing_percent in self.missing_records)
        missing_percent = (missing_percent for taxon, missing_count, missing_percent in self.missing_records)
        self.check_data_type()
        per_taxon_summary = (names, taxa_names, lengths, missing, missing_percent)
        zipped = list(zip(*per_taxon_summary))
        return zipped

    def get_char_summary(self):
        # get summary of frequencies for all characters
        characters = []
        counts = []
        add_to_chars = characters.append
        add_to_counts = counts.append
        char_count_dicts = self.get_counts()
        for char in self.alphabet:
            add_to_chars(char)
            if char in char_count_dicts.keys():
                add_to_counts(str(char_count_dicts[char]))
            else:
                add_to_counts("0")
        return characters, counts

    def get_taxon_char_summary(self):
        # get summary of frequencies for all characters
        records = (self.append_count(char_dict) for taxon, char_dict in self.get_counts_from_parsed())
        return records

    def append_count(self, char_dict):
        count_list = []
        for char in self.alphabet:
            if char in char_dict.keys():
                count_list.append(char_dict[char])
            else:
                count_list.append(0)
        return count_list

    def matrix_creator(self):
        # decompose character matrix into a two-dimensional list
        matrix = [list(sequence) for sequence in self.parsed_aln.values()]
        return matrix

    def get_column(self, i):
        # get site from the character matrix
        return [row[i] for row in self.matrix]

    def all_same(self, site):
        # check if all elements of a site are the same
        return not site or site.count(site[0]) == len(site)

    def get_sites_no_missing_ambiguous(self):
        # get each site without missing or ambiguous characters
        no_missing_ambiguous_sites = [self.get_site_no_missing_ambiguous(column) for column in range(self.get_alignment_length())]
        return no_missing_ambiguous_sites

    def get_site_no_missing_ambiguous(self, column):
        site = self.get_column(column)
        return [char for char in site if char not in self.missing_ambiguous_chars]

    def replace_missing(self, column):
        return ["-" if x in self.missing_chars else x for x in self.get_column(column)]

    def get_trim_selection(self, trim_fraction, parsimony_check):
        # this checks each column of alignment for minimum occupancy
        self.matrix = self.matrix_creator()
        trim_vector = []
        for column in range(self.get_alignment_length()):
            site = self.replace_missing(column)
            occ = (len(site) - site.count("-")) / len(site)
            if parsimony_check:
                unique_chars = set(site)
                try:
                    unique_chars.remove("-")
                except KeyError:
                    pass # this occurs if we have no missing data
                pattern = [base for base in unique_chars if site.count(base) >= 2]
                trim_vector.append(len(pattern) >= 2 and occ >= trim_fraction)
            else:
                trim_vector.append(occ >= trim_fraction)
        return trim_vector

    def get_variable(self):
        # if all elements of a site without missing or ambiguous characters
        # are not the same, consider it variable
        variable = len([site for site in self.no_missing_ambiguous if not self.all_same(site)])
        return variable

    def get_parsimony_informative(self):
        # if the count for a unique character in a site is at least two,
        # and there are at least two such characters in a site without missing
        # or ambiguous characters, consider it parsimony informative
        parsimony_informative = 0
        for site in self.no_missing_ambiguous:
            unique_chars = set(site)
            pattern = [base for base in unique_chars if site.count(base) >= 2]
            no_patterns = len(pattern)

            if no_patterns >= 2:
                parsimony_informative += 1
        return parsimony_informative

    def get_prop_variable(self):
        # get proportion of variable sites to all sites
        prop_variable = self.variable_sites / int(self.length)
        return round(prop_variable, 3)

    def get_prop_parsimony(self):
        # get proportion of parsimony informative sites to all sites
        prop_parsimony = self.parsimony_informative / int(self.length)
        return round(prop_parsimony, 3)

    def get_name(self):
        # get input file name
        in_filename = path.basename(self.in_file)
        return in_filename

    def get_taxa_no(self):
        # get number of taxa
        return len(self.parsed_aln.values())

    def get_alignment_length(self):
        # get alignment length by just checking the first seq length
        # this assumes that all sequences are of equal length
        return len(next(iter(self.parsed_aln.values())))

    def get_matrix_cells(self):
    # count all matrix cells
        self.all_matrix_cells = len(self.parsed_aln.values()) * int(self.length)
        return self.all_matrix_cells

    def get_missing(self):
        # count missing characters from the list of missing for all sequences
        self.missing = sum(count for taxon, count, percent in self.missing_records)
        return self.missing

    def get_missing_percent(self):
        # get missing percent
        missing_percent = round((self.missing / self.all_matrix_cells * 100), 3)
        return missing_percent

    def get_missing_from_parsed(self):
        # get missing count and percent from parsed alignment
        # return a list of tuples with taxon name, count, and percent missing
        self.missing_records = sorted(
            [
                (taxon, self.get_missing_from_seq(seq), self.get_missing_percent_from_seq(seq))
                for taxon, seq in self.parsed_aln.items()
            ]
        )
        return self.missing_records

    def get_missing_from_seq(self, seq):
        # count missing characters for individual sequence
        missing_count = sum(seq.count(char) for char in self.missing_chars)
        return missing_count

    def get_missing_percent_from_seq(self, seq):
        # get missing percent from individual sequence
        missing_seq_percent = round((self.get_missing_from_seq(seq) / self.get_alignment_length() * 100), 3)
        return missing_seq_percent

    def get_counts(self):
        # get counts of each character in the used alphabet for all sequences
        counters = [Counter(chars) for taxon, chars in self.get_counts_from_parsed()]
        all_counts = sum(counters, Counter())
        counts_dict = dict(all_counts)
        return counts_dict

    def get_counts_from_parsed(self):
        # get counts of all characters from parsed alignment
        # return a list of tuples with taxon name and counts
        return sorted(
            [
                (taxon, self.get_counts_from_seq(seq))
                for taxon, seq in self.parsed_aln.items()
            ]
        )

    def get_counts_from_seq(self, seq):
        # get all alphabet chars count for individual sequence
        char_counts = {char : seq.count(char) for char in self.alphabet}
        return char_counts

    def check_data_type(self):
        # check if the data type is correct; only one seq to save on computation
        seq = next(iter(self.parsed_aln.values()))
        self.check = any(char in self.non_alphabet for char in seq)
        if self.check is True:
            print(
                "WARNING: found non-" + self.data_type + " characters. "
                "Are you sure you specified the right data type?"
            )


class AminoAcidAlignment(Alignment):
    """Alphabets specific to amino acid alignments"""

    alphabet = ["A", "C", "D", "E", "F", "G", "H", "I", "K", "L", "M", "N", "P", "Q", "R", "S", "T", "V", "W", "Y", "B", "J", "Z", "X", ".", "*", "-", "?"]
    missing_ambiguous_chars = ["B", "J", "Z", "X", ".", "*", "-", "?"]
    missing_chars = ["X", ".", "*", "-", "?"]
    non_alphabet = ["O"]

    def get_summary(self):
        # get alignment summary specific to amino acids
        data = self.summarize_alignment()
        new_data = data + list(self.get_char_summary()[1])
        return new_data

    def get_taxa_summary(self):
        # get per-taxon/sequence alignment summary specific to amino acids
        data = self.summarize_alignment_by_taxa()
        aa_summary = (data, self.get_taxon_char_summary())
        zipped_list = list(zip(*aa_summary))
        new_data = [list(data_tupl) + chars for data_tupl, chars in zipped_list]
        return new_data

class DNAAlignment(Alignment):
    """Alphabets specific to DNA alignments"""

    alphabet = ["A", "C", "G", "T", "K", "M", "R", "Y", "S", "W", "B", "V", "H", "D", "X", "N", "O", "-", "?"]
    missing_ambiguous_chars = ["K", "M", "R", "Y", "S", "W", "B", "V", "H", "D", "X", "N", "O", "-", "?"]
    missing_chars = ["X", "N", "O", "-", "?"]
    non_alphabet = ["E", "F", "I", "L", "P", "Q", "J", "Z", ".", "*"]

    def get_summary(self):
        # get alignment summarry specific to nucleotide
        data = self.summarize_alignment()
        new_data = data + self.get_atgc_content() + list(self.get_char_summary()[1])
        return new_data

    def get_taxa_summary(self):
        # get per-taxon/sequence alignment summary specific to nucleotides
        data = self.summarize_alignment_by_taxa()
        dna_summary = (data, self.get_list_from_atgc(), self.get_taxon_char_summary())
        zipped_list = list(zip(*dna_summary))
        new_data = [list(data_tupl) + list(atgc) + chars for data_tupl, atgc, chars in zipped_list]
        return new_data

    def get_atgc_content(self):
        # get AC and GC contents for all sequences
        # AT content is the first element of AT, GC content tuple
        # returned by get_atgc_from_seq()
        atgc_records = self.get_atgc_from_parsed()
        at_content = round(sum(atgc[0] for taxon, atgc in atgc_records) / self.get_taxa_no(), 3)
        gc_content = round(1 - float(at_content), 3)

        atgc_content = [str(at_content), str(gc_content)]
        return atgc_content

    def get_list_from_atgc(self):
        records = (atgc for taxon, atgc in self.get_atgc_from_parsed())
        return records

    def get_atgc_from_parsed(self):
        # get AT and GC contents from parsed alignment dictionary
        # return a list of tuples with taxon name, AT content, and GC content
        return sorted([(taxon, self.get_atgc_from_seq(seq)) for taxon, seq in self.parsed_aln.items()])

    def get_atgc_from_seq(self, seq):
        # get AT and GC contents from individual sequences

        at_count = seq.count("A") + seq.count("T") + seq.count("W")
        gc_count = seq.count("G") + seq.count("C") + seq.count("S")

        try:
            at_content = round(at_count / (at_count + gc_count), 3)
            gc_content = round(1 - float(at_content), 3)

        except ZeroDivisionError:
            at_content = 0
            gc_content = 0

        return at_content, gc_content

class MetaAlignment:
    """Class of multiple sequence alignments"""

    def __init__(self, **kwargs):
        # set defaults and get values from kwargs
        self.in_files = kwargs.get("in_files")
        self.in_format = kwargs.get("in_format")
        self.data_type = kwargs.get("data_type")
        self.command = kwargs.get("command")
        self.concat_out = kwargs.get("concat_out", "concatenated.out")
        self.using_metapartitions = False
        self.check_align = kwargs.get("check_align", False)
        self.cores = kwargs.get("cores")
        self.by_taxon_summary = kwargs.get("by_taxon_summary")
        self.no_sup_aln_name = False
        self.no_mpan = False

        if self.command == "concat":
            self.codons = kwargs.get("codons", "none")
            if self.data_type == "aa" and self.codons != "none":
                print("ERROR: when option -d|--data-type is set to ’aa’, option -n|--codons must be set to ’none’.")
                sys.exit(1)

        if self.command == "replicate":
            self.no_replicates = kwargs.get("replicate_args")[0]
            self.no_loci = kwargs.get("replicate_args")[1]

        if self.command == "split":
            self.split = kwargs.get("split_by")
            self.remove_empty = kwargs.get("remove_empty", False)
            self.no_sup_aln_name = kwargs.get("no_sup_aln_name", False)

        if self.command == "metapartitions":
            self.using_metapartitions = True
            self.split = kwargs.get("split_by")
            self.remove_empty = kwargs.get("remove_empty", False)
            self.no_sup_aln_name = kwargs.get("no_sup_aln_name", False)
            self.no_mpan = kwargs.get("no_mpan", False)
            self.prepend_label = kwargs.get("prepend_label")
            if self.prepend_label is not None and isinstance(self.prepend_label, str):
                self.prepend_label = self.prepend_label + "_"
            else:
                self.prepend_label = ""

        if self.command == "remove":
            self.species_to_remove = kwargs.get("taxa_to_remove")
            self.species_to_remove_set = set(self.species_to_remove)
            self.reduced_file_prefix = kwargs.get("out_prefix")
            self.check_taxa = kwargs.get("check_taxa", False)

        if self.command == "translate":
            self.reading_frame = kwargs.get("reading_frame")
            self.genetic_code = kwargs.get("genetic_code")

        if self.command == "trim":
            self.trim_fraction = kwargs.get("trim_fraction")
            self.trim_out = kwargs.get("trim_out")
            self.parsimony_check = kwargs.get("parsimony_check", False)

        self.alignment_objects = self.get_alignment_objects()
        self.parsed_alignments = self.get_parsed_alignments()

        # The code list:
        self.codes_list = """
         1. The Standard Code
         2. The Vertebrate Mitochondrial Code
         3. The Yeast Mitochondrial Code
         4. The Mold, Protozoan, and Coelenterate Mitochondrial Code and the Mycoplasma/Spiroplasma Code
         5. The Invertebrate Mitochondrial Code
         6. The Ciliate, Dasycladacean and Hexamita Nuclear Code
         9. The Echinoderm and Flatworm Mitochondrial Code
        10. The Euplotid Nuclear Code
        11. The Bacterial, Archaeal and Plant Plastid Code
        12. The Alternative Yeast Nuclear Code
        13. The Ascidian Mitochondrial Code
        14. The Alternative Flatworm Mitochondrial Code
        16. Chlorophycean Mitochondrial Code
        21. Trematode Mitochondrial Code
        22. Scenedesmus obliquus Mitochondrial Code
        23. Thraustochytrium Mitochondrial Code
        24. Pterobranchia Mitochondrial Code
        25. Candidate Division SR1 and Gracilibacteria Code
        26. Pachysolen tannophilus Nuclear Code
        """

        # 1: The Standard Code
        self.gencode_NCBI_1 = {
        "TTT" : "F", # Phe
        "TCT" : "S", # Ser
        "TAT" : "Y", # Tyr
        "TGT" : "C", # Cys
        "TTC" : "F", # Phe
        "TCC" : "S", # Ser
        "TAC" : "Y", # Tyr
        "TGC" : "C", # Cys
        "TTA" : "L", # Leu
        "TCA" : "S", # Ser
        "TAA" : "*", # Ter
        "TGA" : "*", # Ter
        "TTG" : "L", # Leu i
        "TCG" : "S", # Ser
        "TAG" : "*", # Ter
        "TGG" : "W", # Trp
        "CTT" : "L", # Leu
        "CCT" : "P", # Pro
        "CAT" : "H", # His
        "CGT" : "R", # Arg
        "CTC" : "L", # Leu
        "CCC" : "P", # Pro
        "CAC" : "H", # His
        "CGC" : "R", # Arg
        "CTA" : "L", # Leu
        "CCA" : "P", # Pro
        "CAA" : "Q", # Gln
        "CGA" : "R", # Arg
        "CTG" : "L", # Leu i
        "CCG" : "P", # Pro
        "CAG" : "Q", # Gln
        "CGG" : "R", # Arg
        "ATT" : "I", # Ile
        "ACT" : "T", # Thr
        "AAT" : "N", # Asn
        "AGT" : "S", # Ser
        "ATC" : "I", # Ile
        "ACC" : "T", # Thr
        "AAC" : "N", # Asn
        "AGC" : "S", # Ser
        "ATA" : "I", # Ile
        "ACA" : "T", # Thr
        "AAA" : "K", # Lys
        "AGA" : "R", # Arg
        "ATG" : "M", # Met i
        "ACG" : "T", # Thr
        "AAG" : "K", # Lys
        "AGG" : "R", # Arg
        "GTT" : "V", # Val
        "GCT" : "A", # Ala
        "GAT" : "D", # Asp
        "GGT" : "G", # Gly
        "GTC" : "V", # Val
        "GCC" : "A", # Ala
        "GAC" : "D", # Asp
        "GGC" : "G", # Gly
        "GTA" : "V", # Val
        "GCA" : "A", # Ala
        "GAA" : "E", # Glu
        "GGA" : "G", # Gly
        "GTG" : "V", # Val
        "GCG" : "A", # Ala
        "GAG" : "E", # Glu
        "GGG" : "G", # Gly
        "---" : "-", # Gap
        "???" : "?", # Unk
        "NNN" : "X", # Unk
        }

        # 2: The Vertebrate Mitochondrial Code
        self.gencode_NCBI_2 = self.gencode_NCBI_1.copy()
        self.gencode_NCBI_2["AGA"] = "*" # Ter
        self.gencode_NCBI_2["AGG"] = "*" # Ter
        self.gencode_NCBI_2["ATA"] = "M" # Met
        self.gencode_NCBI_2["TGA"] = "W" # Trp

        # 3: The Yeast Mitochondrial Code
        self.gencode_NCBI_3 = self.gencode_NCBI_1.copy()
        self.gencode_NCBI_3["ATA"] = "M" # Met
        self.gencode_NCBI_3["CTT"] = "T" # Thr
        self.gencode_NCBI_3["CTC"] = "T" # Thr
        self.gencode_NCBI_3["CTA"] = "T" # Thr
        self.gencode_NCBI_3["CTG"] = "T" # Thr
        self.gencode_NCBI_3["TGA"] = "W" # Trp

        del self.gencode_NCBI_3["CGA"]
        del self.gencode_NCBI_3["CGC"]

        # 4: The Mold, Protozoan, and Coelenterate Mitochondrial Code and the Mycoplasma/Spiroplasma Code
        self.gencode_NCBI_4 = self.gencode_NCBI_1.copy()
        self.gencode_NCBI_4["TGA"] = "W" # Trp

        # 5: The Invertebrate Mitochondrial Code
        self.gencode_NCBI_5 = self.gencode_NCBI_1.copy()
        self.gencode_NCBI_5["AGA"] = "S" # Ser
        self.gencode_NCBI_5["AGG"] = "S" # Ser
        self.gencode_NCBI_5["ATA"] = "M" # Met
        self.gencode_NCBI_5["TGA"] = "W" # Trp

        # 6: The Ciliate, Dasycladacean and Hexamita Nuclear Code
        self.gencode_NCBI_6 = self.gencode_NCBI_1.copy()
        self.gencode_NCBI_6["TAA"] = "Q" # Gln
        self.gencode_NCBI_6["TAG"] = "Q" # Gln

        # 9: The Echinoderm and Flatworm Mitochondrial Code
        self.gencode_NCBI_9 = self.gencode_NCBI_1.copy()
        self.gencode_NCBI_9["AAA"] = "N" # Asn
        self.gencode_NCBI_9["AGA"] = "S" # Ser
        self.gencode_NCBI_9["AGG"] = "S" # Ser
        self.gencode_NCBI_9["TGA"] = "W" # Trp

        # 10: The Euplotid Nuclear Code
        self.gencode_NCBI_10 = self.gencode_NCBI_1.copy()
        self.gencode_NCBI_10["TGA"] = "C" # Cys

        # 11: The Bacterial, Archaeal and Plant Plastid Code
        self.gencode_NCBI_11 = self.gencode_NCBI_1.copy()

        # 12: The Alternative Yeast Nuclear Code
        self.gencode_NCBI_12 = self.gencode_NCBI_1.copy()
        self.gencode_NCBI_12["CTG"] = "S" # Ser

        # 13: The Ascidian Mitochondrial Code
        self.gencode_NCBI_13 = self.gencode_NCBI_1.copy()
        self.gencode_NCBI_13["AGA"] = "G" # Gly
        self.gencode_NCBI_13["AGG"] = "G" # Gly
        self.gencode_NCBI_13["ATA"] = "M" # Met
        self.gencode_NCBI_13["TGA"] = "W" # Trp

        # 14: The Alternative Flatworm Mitochondrial Code
        self.gencode_NCBI_14 = self.gencode_NCBI_1.copy()
        self.gencode_NCBI_14["AAA"] = "N" # Asn
        self.gencode_NCBI_14["AGA"] = "S" # Ser
        self.gencode_NCBI_14["AGG"] = "S" # Ser
        self.gencode_NCBI_14["TAA"] = "Y" # Tyr
        self.gencode_NCBI_14["TGA"] = "W" # Trp

        # 16: Chlorophycean Mitochondrial Code
        self.gencode_NCBI_16 = self.gencode_NCBI_1.copy()
        self.gencode_NCBI_16["TAG"] = "L" # Leu

        # 21: Trematode Mitochondrial Code
        self.gencode_NCBI_21 = self.gencode_NCBI_1.copy()
        self.gencode_NCBI_21["TGA"] = "W" # Trp
        self.gencode_NCBI_21["ATA"] = "M" # Met
        self.gencode_NCBI_21["AGA"] = "S" # Ser
        self.gencode_NCBI_21["AGG"] = "S" # Ser
        self.gencode_NCBI_21["AAA"] = "N" # Asn

        # 22: Scenedesmus obliquus Mitochondrial Code
        self.gencode_NCBI_22 = self.gencode_NCBI_1.copy()
        self.gencode_NCBI_22["TCA"] = "*" # Ter
        self.gencode_NCBI_22["TAG"] = "L" # Leu

        # 23: Thraustochytrium Mitochondrial Code
        self.gencode_NCBI_23 = self.gencode_NCBI_1.copy()
        self.gencode_NCBI_23["TTA"] = "*" # Ter

        # 24: Pterobranchia Mitochondrial Code
        self.gencode_NCBI_24 = self.gencode_NCBI_1.copy()
        self.gencode_NCBI_24["AGA"] = "S" # Ser
        self.gencode_NCBI_24["AGG"] = "K" # Lys
        self.gencode_NCBI_24["TGA"] = "W" # Trp

        # 25: Candidate Division SR1 and Gracilibacteria Code
        self.gencode_NCBI_25 = self.gencode_NCBI_1.copy()
        self.gencode_NCBI_25["TGA"] = "G" # Gly

        # 26: Pachysolen tannophilus Nuclear Code
        self.gencode_NCBI_26 = self.gencode_NCBI_1.copy()
        self.gencode_NCBI_26["CTG"] = "A" # Ala

        self.codes = {
        1 : self.gencode_NCBI_1,
        2 : self.gencode_NCBI_2,
        3 : self.gencode_NCBI_3,
        4 : self.gencode_NCBI_4,
        5 : self.gencode_NCBI_5,
        6 : self.gencode_NCBI_6,
        9 : self.gencode_NCBI_9,
        10 : self.gencode_NCBI_10,
        11 : self.gencode_NCBI_11,
        12 : self.gencode_NCBI_12,
        13 : self.gencode_NCBI_13,
        14 : self.gencode_NCBI_14,
        16 : self.gencode_NCBI_16,
        21 : self.gencode_NCBI_21,
        22 : self.gencode_NCBI_22,
        23 : self.gencode_NCBI_23,
        24 : self.gencode_NCBI_24,
        25 : self.gencode_NCBI_25,
        26 : self.gencode_NCBI_26
        }

    def translate_dna_to_aa(self, seq, translation_table, frame):
        # translate DNA string into amino acids
        # where the last codon starts
        last_codon_start = len(seq) - 2
        # where the first codon starts
        if frame == 1:
            first = 0
        elif frame == 2:
            first = 1
        elif frame == 3:
            first = 2
        # create protein sequence by growing list
        protein = []
        add_to_protein = protein.append
        for start in range(first, last_codon_start, 3):
            codon = seq[start : start + 3]
            aa = translation_table.get(codon.upper(), 'X')
            add_to_protein(aa)

        return "".join(protein)

    def translate_dict(self, source_dict):
        translation_table = self.codes.get(self.genetic_code)
        translated_dict = {}
        for taxon, seq in sorted(source_dict.items()):
            translated_seq = self.translate_dna_to_aa(seq, translation_table, self.reading_frame)
            if "*" in translated_seq:
                print("WARNING: stop codon(s), indicated as *, found in {} sequence".format(taxon))
            translated_dict[taxon] = translated_seq

        return translated_dict

    def get_translated(self, translation_table, reading_frame):
        if int(self.cores) == 1:
            translated_alignments = [self.translate_dict(alignment) for alignment in self.parsed_alignments]
        elif int(self.cores) > 1:
            pool = mp.Pool(int(self.cores))
            translated_alignments = pool.map(self.translate_dict, self.parsed_alignments)

        return translated_alignments

    def trim_dict(self, alignment):
        trim_vector = alignment.get_trim_selection(self.trim_fraction, self.parsimony_check)
        aln_dict = alignment.parsed_aln
        for key in aln_dict:
            aln_dict[key] = ”.join(list(compress(aln_dict[key], trim_vector)))

        return aln_dict

    def get_trimmed(self, trim_fraction, parsimony_check):
        if int(self.cores) == 1:
            trimmed_alignments = [self.trim_dict(alignment) for alignment in self.alignment_objects]
        elif int(self.cores) > 1:
            pool = mp.Pool(int(self.cores))
            trimmed_alignments = pool.map(self.trim_dict, self.alignment_objects)

        return trimmed_alignments

    def remove_unknown_chars(self, seq):
        # remove unknown characters from sequence
        new_seq = seq.replace("?", "").replace("-", "")

        return new_seq

    def remove_empty_sequences(self, split_alignment):
        # remove taxa from alignment if they are composed of only empty sequences
        new_alignment = {taxon : seq for taxon, seq in split_alignment.items() if
self.remove_unknown_chars(seq)}

        return new_alignment

    def get_partitions(self, partitions_file):
        # parse and get partitions from partitions file
        partitions = FileParser(partitions_file)
        parsed_partitions = partitions.partitions_parse()

        return parsed_partitions

    def get_alignment_object(self, alignment):
        # parse according to the given alphabet;
        # Note:(‘alignment‘) <=> `in_file‘ outside MetaAlignment, e.g.
        #
AminoAcidAlignment(Alignment<-.get_parsed_aln<-.get_aln_input)<-FileParser.__init__(in_file)<-FileHandler(...open(self.
        if self.data_type == "aa":
            aln = AminoAcidAlignment(alignment, self.in_format, self.data_type)
        elif self.data_type == "dna":
            aln = DNAAlignment(alignment, self.in_format, self.data_type)
        return aln

    def get_alignment_objects(self):
        # get alignment objects on which statistics can be computed
        # use multiprocessing if more than one core specified
        if int(self.cores) == 1:
            alignments = [self.get_alignment_object(alignment) for alignment in self.in_files]
        elif int(self.cores) > 1:
            pool = mp.Pool(int(self.cores))
            alignments = pool.map(self.get_alignment_object, self.in_files)
        return alignments

    def get_parsed_alignments(self):
        # get parsed dictionaries with taxa and sequences
        parsed_alignments = []
        add_to_parsed_alignments = parsed_alignments.append
        for alignment in self.alignment_objects:
            parsed = alignment.parsed_aln
            add_to_parsed_alignments(parsed)
            # checking if every seq has the same length or if parsed is not empty; exit if false
            if self.check_align:
                equal = all(
                    x == [len(list(parsed.values())[i]) for i in
range(0,len(list(parsed.values())))][0]
                    for x in [len(list(parsed.values())[i]) for i in
range(0,len(list(parsed.values())))]
                )
                if equal is False:
                    print("ERROR: Sequences in input are of varying lengths. Be sure to align them
first.")
                    sys.exit()

            if not parsed.keys() or not any(parsed.values()):
                print(
                    "ERROR: Parsed sequences of " + alignment.in_file + " are empty. "
                    "Are you sure you specified the right input format and/or that input is a valid
alignment?"
                )
                sys.exit()

        return parsed_alignments

    def get_partitioned(self, partitions_file):
        # partition alignment according to a partitions file
        partitions = self.get_partitions(partitions_file)
        alignment = self.parsed_alignments[0]

        # initiate list of newly partitioned alignments
        list_of_parts = []
        add_to_list_of_parts = list_of_parts.append
        for partition in partitions:
            # loop over all parsed partitions, adding taxa and sliced sequences
            for name, elements in partition.items():
                new_dict = {}

                for taxon, seq in alignment.items():
                    new_seq = ""

                    for dictionary in elements:
                        new_seq = new_seq +
seq[dictionary["start"]:dictionary["stop"]:dictionary["stride"]]
                        new_dict[taxon] = new_seq

                if self.remove_empty:
                    # check if remove empty sequences
                    no_empty_dict = self.remove_empty_sequences(new_dict)
                    add_to_list_of_parts({name : no_empty_dict})
                else:
                    # add partition name : dict of taxa and sequences to the list
                    add_to_list_of_parts({name : new_dict})

        return list_of_parts

    def get_summaries(self):
        # get summaries for all alignment objects

        # define different headers for aa and dna alignments
        aa_header = [
            "Alignment_name",
            "No_of_taxa",
            "Alignment_length",
            "Total_matrix_cells",
            "Undetermined_characters",
            "Missing_percent",
            "No_variable_sites",
            "Proportion_variable_sites",
            "Parsimony_informative_sites",
            "Proportion_parsimony_informative"
        ]

        dna_header = [
            "Alignment_name",
            "No_of_taxa",
            "Alignment_length",
            "Total_matrix_cells",
            "Undetermined_characters",
            "Missing_percent",
            "No_variable_sites",
            "Proportion_variable_sites",
            "Parsimony_informative_sites",
            "Proportion_parsimony_informative",
            "AT_content",
            "GC_content"
        ]

        alignments = self.alignment_objects
        parsed_alignments = self.parsed_alignments
        freq_header = [char for char in alignments[0].alphabet]

        if self.data_type == "aa":
            header = aa_header + freq_header
        elif self.data_type == "dna":
            header = dna_header + freq_header

        # use multiprocessing if more than one core specified
        if int(self.cores) == 1:
            summaries = [alignment.get_summary() for alignment in alignments]
        elif int(self.cores) > 1:
            pool = mp.Pool(int(self.cores))
            summaries = pool.map(self.summarize_alignments, alignments)
        return header, summaries

    def summarize_alignments(self, alignment):
        # helper function to summarize alignments
        summary = alignment.get_summary()
        return summary

    def get_taxon_summaries(self):
        # get per-sequence summaries for all alignment objects

        # define different headers for aa and dna alignments
        aa_header = [
            "Alignment_name",
            "Taxon_name",
            "Sequence_length",
            "Undetermined_characters",
            "Missing_percent"
        ]

        dna_header = [
            "Alignment_name",
            "Taxon_name",
            "Sequence_length",
            "Undetermined_characters",
            "Missing_percent",
            "AT_content",
            "GC_content"
        ]

        alignments = self.alignment_objects
        parsed_alignments = self.parsed_alignments
        freq_header = alignments[0].alphabet

        if self.data_type == "aa":
            header = aa_header + freq_header
        elif self.data_type == "dna":
            header = dna_header + freq_header

        # use multiprocessing if more than one core specified
        if int(self.cores) == 1:
            summaries = [alignment.get_taxa_summary() for alignment in alignments]
        elif int(self.cores) > 1:
            pool = mp.Pool(int(self.cores))
            summaries = pool.map(self.summarize_alignments_taxa, alignments)

        return header, summaries

    def summarize_alignments_taxa(self, alignment):
        # helper function to summarize alignments by taxon
        summary = alignment.get_taxa_summary()
        return summary

    def write_summaries(self, file_name):
        # write summaries to file

        self.file_overwrite_error(file_name)

        with open(file_name, "w", encoding="utf-8") as summary_file:
            summary_out = self.get_summaries()
            header = '\t'.join(summary_out[0])
            new_summ = ['\t'.join(summary) for summary in summary_out[1]]
            summary_file.write(header + '\n')
            summary_file.write('\n'.join(new_summ))
            summary_file.write('\n')
            print("Wrote summaries to file '" + file_name + "'")

    def write_taxa_summaries(self):
        # write by-taxon summaries to file
        for index, in_file_name in enumerate(self.in_files):
            out_file_name = in_file_name + "-seq-summary.txt"
            self.file_overwrite_error(out_file_name)
            with open(out_file_name, "w", encoding="utf-8") as summary_file:
                summary_out = self.get_taxon_summaries()
                header = '\t'.join(summary_out[0])
                summ = [[str(col) for col in element] for element in summary_out[1][index]]
                new_summ = ['\t'.join(row) for row in summ]
                summary_file.write(header + '\n')
                summary_file.write('\n'.join(new_summ))
                summary_file.write('\n')

    def get_replicate(self, no_replicates, no_loci):
        # construct replicate data sets for phylogenetic jackknife
        replicates = []
        add_to_replicates = replicates.append
        counter = 1
        for replicate in range(no_replicates):

            try:
                random_alignments = sample(self.parsed_alignments, no_loci)
            except ValueError:
                print("ERROR: You specified more loci per replicate than there are in your input.")
                sys.exit()

            random_alignments = sample(self.parsed_alignments, no_loci)
            concat_replicate = self.get_concatenated(random_alignments)[0]
            add_to_replicates(concat_replicate)
            counter += 1

        return replicates

    def get_concatenated(self, alignments):
        # concatenate muntiple input alignments
        # create empty dictionary of lists
        concatenated = defaultdict(list)

        # first create list of taxa in all alignments
        # you need this to insert empty seqs in
        # the concatenated alignment
        all_taxa = []
        for alignment in alignments:
            for taxon in alignment.keys():
                if taxon not in all_taxa:
                    all_taxa.append(taxon)

        # start counters to keep track of partitions
        partition_counter = 1
        position_counter = 1
        # get dict for alignment name and partition
        partitions = {}
        digits_to_pad = len(str(len(alignments)))

        for alignment in alignments:
            # get alignment length from a random taxon
            partition_length = len(alignment[list(alignment.keys())[0]])
            # get base name of each alignment for use when writing partitions file
            # NOTE: the base name here is whatever comes before fist period in the file name
            alignment_name = self.alignment_objects[partition_counter - 1].get_name().split(’.’)[0]

            if self.using_metapartitions:
                # implementation of option --no-mpan; option --prepend(-label) will assign a string or "" (see class definition)
                if self.no_mpan:
                    # omit original alignment names from the printed partition file
                    partition_name = self.prepend_label + "p" + str(partition_counter).zfill(digits_to_pad)
                else:
                    # keep original alignment names in the printed partition file
                    partition_name = self.prepend_label + "p" + str(partition_counter).zfill(digits_to_pad) + "_" + alignment_name
            else:
                partition_name = "p" + str(partition_counter) + "_" + alignment_name

            start = position_counter
            position_counter += partition_length
            end = position_counter - 1
            partitions[partition_name] = str(start) + "-" + str(end)
            partition_counter += 1

            # get empty sequence if there is missing taxon
            # getting length from first element of list of keys
            # created from the original dict for this alignment
            empty_seq = ’?’ * partition_length

            for taxon in all_taxa:

                if taxon not in alignment.keys():
                    concatenated[taxon].append(empty_seq)
                else:
                    concatenated[taxon].append(alignment[taxon])

        concatenated = {taxon:”.join(seqs) for taxon, seqs in concatenated.items()}

        return concatenated, partitions

    def remove_from_alignment(self, alignment, species_to_remove_set, index):
        # remove taxa from alignment
        aln_name = self.get_alignment_name_no_ext(index)
        for taxon in species_to_remove_set:
            if taxon not in alignment.keys():
                print(
                    "WARNING: Taxon ’" + taxon + "’ not found in ’" + aln_name + "’.\nIf you expected it to be there, "
                    "make sure to replace all taxon name spaces with underscores and that you are not using quotes."
                )
        # originally within for-loop scope (redundancy)
        new_alignment = {species: seq for species, seq in alignment.items() if species not in species_to_remove_set}

        return aln_name, new_alignment

    def remove_taxa(self, species_to_remove_set):
        new_alns = {}
        for index, alignment in enumerate(self.parsed_alignments):
            aln_name, aln_dict = self.remove_from_alignment(alignment, species_to_remove_set, index)
            # check if alignment is not empty:
            if aln_dict:
                new_alns[aln_name] = aln_dict
            else:
                print("ERROR: You asked to remove all taxa from the alignment " + aln_name + ". No output file will be written.")

        return new_alns

    def print_fasta(self, source_dict):
        # print fasta-formatted string from a dictionary
        fasta_string = ""
        # each sequence line will have 80 characters
        n = 80

        for taxon, seq in sorted(source_dict.items()):
            # split dictionary values to a list of string, each n chars long
            seq = [seq[i:i+n] for i in range(0, len(seq), n)]
            # in case there are unwanted spaces in taxon names
            taxon = taxon.replace(" ", "_").strip("’")
            fasta_string += ">" + taxon + "\n"
            for element in seq:
                fasta_string += element + "\n"

        return fasta_string

    def print_phylip(self, source_dict):
        # print phylip-formatted string from a dictionary
        taxa_list = list(source_dict.keys())
        no_taxa = len(taxa_list)
        # figure out the max length of a taxon for nice padding of sequences
        pad_longest_name = len(max(taxa_list, key=len)) + 3
        # get sequence length from a random value
        seq_length = len(next(iter(source_dict.values())))
        header = str(len(source_dict)) + " " + str(seq_length)
        phylip_string = header + "\n"
        for taxon, seq in sorted(source_dict.items()):
            taxon = taxon.replace(" ", "_").strip("’")
            # left-justify taxon names relative to sequences
            phylip_string += taxon.ljust(pad_longest_name, ’ ’) + seq + "\n"

        return phylip_string

    def print_phylip_int(self, source_dict):
        # print phylip interleaved-formatted string from a dictionary
        taxa_list = list(source_dict.keys())
        no_taxa = len(taxa_list)
        pad_longest_name = len(max(taxa_list, key=len)) + 3
        seq_length = len(next(iter(source_dict.values())))
        header = str(len(source_dict)) + " " + str(seq_length)
        phylip_int_string = header + "\n\n"
        # this will be a list of tuples to hold taxa names and sequences
        seq_matrix = []

        # each sequence line will have 500 characters
        n = 500

        # recreate sequence matrix
        add_to_matrix = seq_matrix.append
        for taxon, seq in sorted(source_dict.items()):
            add_to_matrix((taxon, [seq[i:i+n] for i in range(0, len(seq), n)]))

        first_seq = seq_matrix[0][1]
        for index, item in enumerate(first_seq):
            for taxon, sequence in seq_matrix:
                if index == 0:
                    phylip_int_string += taxon.ljust(pad_longest_name, ’ ’) + sequence[index] + "\n"
                else:
                    phylip_int_string += sequence[index] + "\n"
            phylip_int_string += "\n"

        return phylip_int_string

    def print_nexus(self, source_dict):
        # print nexus-formatted string from a dictionary
        if self.data_type == "aa" or self.command == "translate":
            data_type = "PROTEIN"
        elif self.data_type == "dna":
            data_type = "DNA"

        taxa_list = list(source_dict.keys())
        no_taxa = len(taxa_list)
        pad_longest_name = len(max(taxa_list, key=len)) + 3
        seq_length = len(next(iter(source_dict.values())))
        header = str(len(source_dict)) + " " + str(seq_length)
        nexus_string = (
            "#NEXUS\n\nBEGIN DATA;\n\tDIMENSIONS  NTAX=" + str(no_taxa) + " NCHAR=" + str(seq_length)
            + ";\n\tFORMAT DATATYPE=" + data_type + "  GAP = - MISSING = ?;\n\tMATRIX\n"
        )

        for taxon, seq in sorted(source_dict.items()):
            taxon = taxon.replace(" ", "_").strip("’")
            nexus_string += "\t" + taxon.ljust(pad_longest_name, ’ ’) + seq + "\n"
        nexus_string += "\n;\n\nEND;"

        return nexus_string

    def print_nexus_int(self, source_dict):
        # print nexus interleaved-formatted string from a dictionary
        if self.data_type == "aa":
            data_type = "PROTEIN"
        elif self.data_type == "dna":
            data_type = "DNA"

        taxa_list = list(source_dict.keys())
        no_taxa = len(taxa_list)
        pad_longest_name = len(max(taxa_list, key=len)) + 3
        seq_length = len(next(iter(source_dict.values())))
        header = str(len(source_dict)) + " " + str(seq_length)
        # this will be a list of tuples to hold taxa names and sequences
        seq_matrix = []
        nexus_int_string = (
            "#NEXUS\n\nBEGIN DATA;\n\tDIMENSIONS  NTAX=" + str(no_taxa) + " NCHAR=" + str(seq_length)
            + ";\n\tFORMAT   INTERLEAVE" + "   DATATYPE=" + data_type + "  GAP = - MISSING = ?;\n\tMATRIX\n"
        )
        # each sequence line will have 500 characters
        n = 500

        # recreate sequence matrix
        add_to_matrix = seq_matrix.append
        for taxon, seq in sorted(source_dict.items()):
            add_to_matrix((taxon, [seq[i:i+n] for i in range(0, len(seq), n)]))

        first_seq = seq_matrix[0][1]
        for index, item in enumerate(first_seq):
            for taxon, sequence in seq_matrix:
                if index == 0:
                    nexus_int_string += taxon.ljust(pad_longest_name, ’ ’) + sequence[index] + "\n"
                else:
                    nexus_int_string += sequence[index] + "\n"
            nexus_int_string += "\n"

        nexus_int_string += "\n;\n\nEND;"

        return nexus_int_string

    def natural_sort(self, a_list):
        # create a function that does ’human sort’ on a list
        convert = lambda text: int(text) if text.isdigit() else text.lower()
        alphanum_key = lambda key: [convert(c) for c in re.split(’([0-9]+)’, key)]
        return sorted(a_list, key = alphanum_key)

    def print_unspecified_partitions(self, data_type, codons):
        # print partitions for concatenated alignment
        part_string = ""
        part_dict = self.get_concatenated(self.parsed_alignments)[1]
        part_list = self.natural_sort(part_dict.keys())

        if data_type == "dna":
            if codons == "none":
                for key in part_list:
                    part_string += key + " = " + str(part_dict[key]) + "\n"
            elif codons == "12":
                for key in part_list:
                    start, end = str(part_dict[key]).split("-")
                    part_string += key + "_pos1" + " = " + start + "-" + end + "\\2" + "\n"
                    part_string += key + "_pos2" + " = " + str(int(start) + 1) + "-" + end + "\\2" + "\n"
            elif codons == "123":
                for key in part_list:
                    start, end = str(part_dict[key]).split("-")
                    part_string += key + "_pos1" + " = " + start + "-" + end + "\\3" + "\n"
                    part_string += key + "_pos2" + " = " + str(int(start) + 1) + "-" + end + "\\3" + "\n"
                    part_string += key + "_pos3" + " = " + str(int(start) + 2) + "-" + end + "\\3" + "\n"

        elif data_type == "aa":
            for key in part_list:
                part_string += key + " = " + str(part_dict[key]) + "\n"

        return part_string
    def print_nexus_partitions(self, data_type, codons):
        # print partitions for concatenated alignment
        part_string = ""
        part_dict = self.get_concatenated(self.parsed_alignments)[1]
        part_list = self.natural_sort(part_dict.keys())
        # write beginning of nexus sets
        part_string += "#NEXUS\n\n"
        part_string += "BEGIN SETS;\n"

        if data_type == "dna":
            if codons == "none":
                for key in part_list:
                    part_string += "\tcharset " + key + " = " + str(part_dict[key]) + ";\n"
            elif codons == "12":
                for key in part_list:
                    start, end = str(part_dict[key]).split("-")
                    part_string += "\tcharset " + key + "_pos1" + " = " + start + "-" + end + "\\2" + ";\n"
                    part_string += "\tcharset " + key + "_pos2" + " = " + str(int(start) + 1) + "-" + end + "\\2" + ";\n"
            elif codons == "123":
                for key in part_list:
                    start, end = str(part_dict[key]).split("-")
                    part_string += "\tcharset " + key + "_pos1" + " = " + start + "-" + end + "\\3" + ";\n"
                    part_string += "\tcharset " + key + "_pos2" + " = " + str(int(start) + 1) + "-" + end + "\\3" + ";\n"
                    part_string += "\tcharset " + key + "_pos3" + " = " + str(int(start) + 2) + "-" + end + "\\3" + ";\n"
            part_string += "END;"

        elif data_type == "aa":
            for key in part_list:
                part_string += "\tcharset " + key + " = " + str(part_dict[key]) + ";\n"
            part_string += "END;"

        return part_string

    def print_iqtree_nexus_partitions(self, data_type, codons):
        # print partitions for concatenated alignment
        part_string = ""
        part_dict = self.get_concatenated(self.parsed_alignments)[1]
        part_list = self.natural_sort(part_dict.keys())
        # write beginning of nexus sets
        part_string += "#nexus\n"
        part_string += "begin sets;\n"

        if data_type == "dna":
            if codons == "none":
                for key in part_list:
                    part_string += "  charset " + key + " = " + str(part_dict[key]) + ";\n"
            elif codons == "12":
                for key in part_list:
                    start, end = str(part_dict[key]).split("-")
                    part_string += "  charset " + key + "_pos1" + " = " + start + " - " + end + "\\2" + ";\n"
                    part_string += "  charset " + key + "_pos2" + " = " + str(int(start) + 1) + " - " + end + "\\2" + ";\n"
            elif codons == "123":
                for key in part_list:
                    start, end = str(part_dict[key]).split("-")
                    part_string += "  charset " + key + "_pos1" + " = " + start + " - " + end + "\\3" + ";\n"
                    part_string += "  charset " + key + "_pos2" + " = " + str(int(start) + 1) + " - " + end + "\\3" + ";\n"
                    part_string += "  charset " + key + "_pos3" + " = " + str(int(start) + 2) + " - " + end + "\\3" + ";\n"
            part_string += "end;\n"

        elif data_type == "aa":
            for key in part_list:
                part_string += "  charset " + key + " = " + str(part_dict[key]) + ";\n"
            part_string += "end;\n"

        return part_string

    def print_raxml_partitions(self, data_type, codons):
        # print partitions for concatenated alignment
        part_string = ""
        part_dict = self.get_concatenated(self.parsed_alignments)[1]
        part_list = self.natural_sort(part_dict.keys())

        if data_type == "dna":
            if codons == "none":
                for key in part_list:
                    part_string += "DNA, " + key + " = " + str(part_dict[key]) + "\n"
            elif codons == "12":
                for key in part_list:
                    start, end = str(part_dict[key]).split("-")
                    part_string += "DNA, " + key + "_pos1" + " = " + start + "-" + end + "\\2" + "\n"
                    part_string += "DNA, " + key + "_pos2" + " = " + str(int(start) + 1) + "-" + end + "\\2" + "\n"
            elif codons == "123":
                for key in part_list:
                    start, end = str(part_dict[key]).split("-")
                    part_string += "DNA, " + key + "_pos1" + " = " + start + "-" + end + "\\3" + "\n"
                    part_string += "DNA, " + key + "_pos2" + " = " + str(int(start) + 1) + "-" + end + "\\3" + "\n"
                    part_string += "DNA, " + key + "_pos3" + " = " + str(int(start) + 2) + "-" + end + "\\3" + "\n"

        elif data_type == "aa":
            for key in part_list:
                part_string += "WAG, " + key + " = " + str(part_dict[key]) + "\n"

            #  aa-partition files with strides are probably not useful? (original below)
#            elif codons == "12":
#                for key in part_list:
#                    start, end = str(part_dict[key]).split("-")
#                    part_string += "WAG, " + key + "_pos1" + " = " + start + "-" + end + "\\2" + "\n"
#                    part_string += "WAG, " + key + "_pos2" + " = " + str(int(start) + 1) + "-" + end + "\\2" + "\n"
#            elif codons == "123":
#                for key in part_list:
#                    start, end = str(part_dict[key]).split("-")
#                    part_string += "WAG, " + key + "_pos1" + " = " + start + "-" + end + "\\3" + "\n"
#                    part_string += "WAG, " + key + "_pos2" + " = " + str(int(start) + 1) + "-" + end + "\\3" + "\n"
#                    part_string += "WAG, " + key + "_pos3" + " = " + str(int(start) + 2) + "-" + end + "\\3" + "\n"
        return part_string

    def replace_string_in_file(self, file_name, old_string, new_string):
        # global string replacement in file
        with open(file_name, "r", encoding="utf-8") as file:
            file_content = file.read()
        # write globally replaced content back to file
        glb_replaced_content = file_content.replace(old_string, new_string)
        with open(file_name, "w", encoding="utf-8") as file:
            file.write(glb_replaced_content)

    def write_partitions(self, file_name, part_format, data_type, codons):
        # write partitions file for concatenated alignment
        self.file_overwrite_error(file_name)
        with open(file_name, "w", encoding="utf-8") as part_file:
            if part_format == "nexus":
                part_file.write(self.print_nexus_partitions(data_type, codons))
            if part_format == "iqtree-nexus":
                part_file.write(self.print_iqtree_nexus_partitions(data_type, codons))
            if part_format == "raxml":
                part_file.write(self.print_raxml_partitions(data_type, codons))
            if part_format == "unspecified":
                part_file.write(self.print_unspecified_partitions(data_type, codons))

            if self.using_metapartitions:
                self.replace_string_in_file(file_name, '-meta =', ' =')

        print("Wrote partitions for the concatenated file to '" + file_name + "'")

    def get_extension(self, file_format):
        # get proper extension string
        if file_format == "phylip":
            extension = "-out.phy"
        elif file_format == "phylip-int":
            extension = "-out.int-phy"
        elif file_format == "fasta":
            extension = "-out.fas"
        elif file_format == "nexus":
            extension = "-out.nex"
        elif file_format == "nexus-int":
            extension = "-out.int-nex"

        return extension

    def get_metapartition_extension(self, file_format):
        # get proper metapartition_extension string
        if file_format == "phylip":
            metapartition_extension = "-meta.phy"
        elif file_format == "phylip-int":
            metapartition_extension = "-meta.int-phy"
        elif file_format == "fasta":
            metapartition_extension = "-meta.fas"
        elif file_format == "nexus":
            metapartition_extension = "-meta.nex"
        elif file_format == "nexus-int":
            metapartition_extension = "-meta.int-nex"

        return metapartition_extension

    def file_overwrite_error(self, file_name):
        # print warning when overwriting a file
        if path.exists(file_name):
            print("WARNING: You are overwriting ’" + file_name + "’")

    def write_formatted_file(self, file_format, file_name, alignment):
        # write the correct format string into a file
        with open(file_name, "w", encoding="utf-8") as out_file:
            if file_format == "phylip":
                out_file.write(self.print_phylip(alignment))
            elif file_format == "fasta":
                out_file.write(self.print_fasta(alignment))
            elif file_format == "phylip-int":
                out_file.write(self.print_phylip_int(alignment))
            elif file_format == "nexus":
                out_file.write(self.print_nexus(alignment))
            elif file_format == "nexus-int":
                out_file.write(self.print_nexus_int(alignment))

    def get_alignment_name(self, i, extension):
        # get file name
        file_name = self.alignment_objects[i].get_name() + extension

        return file_name

    def get_alignment_name_no_ext(self, i):
        # get file name without extension
        file_name = self.alignment_objects[i].get_name()

        return file_name

    def write_concat(self, file_format):
        # write concatenated alignment into a file
        concatenated_alignment = self.get_concatenated(self.parsed_alignments)[0]
        file_name = self.concat_out
        self.file_overwrite_error(file_name)
        self.write_formatted_file(file_format, file_name, concatenated_alignment)

        print("Wrote concatenated sequences to " + file_format + " file ’" + file_name + "’")

    def write_convert(self, index, alignment, file_format, extension):
        # write converted alignment into a file
        file_name = self.get_alignment_name(index, extension)
        self.file_overwrite_error(file_name)
        self.write_formatted_file(file_format, file_name, alignment)

    def write_replicate(self, index, alignment, file_format, extension):
        # write replicate alignment into a file
        file_name = "replicate" + str(index + 1) + "_" + str(self.no_loci) + "-loci" + extension
        self.file_overwrite_error(file_name)
        self.write_formatted_file(file_format, file_name, alignment)

    def write_split(self, item, file_format, extension):
        # write split alignments from partitions file
        # bad practice with the dicts; figure out better solution
        partition_name = list(item.keys())[0]
        alignment = item[partition_name]

        if not alignment:
            # If the alignment dict is empty, i.e. no alignment associated with partition name, raise error
            raise ValueError("Partition ’%s’ is empty. No sequences to write." % partition_name)

        # implementation of option --no-san (don’t prepend input superalignment filename to the `split` outputs)
        if self.no_sup_aln_name:
            file_name = partition_name + extension
        else:
            file_name = str(self.in_files[0].split(’.’)[0]) + "_" + partition_name + extension

        try:
            self.file_overwrite_error(file_name)
            self.write_formatted_file(file_format, file_name, alignment)
            yield file_name
        except ValueError as e:
            print("There was an issue writing file ’%s’: %s" % (file_name, str(e)))
            remove(file_name)
            raise

    def write_reduced(self, file_format, extension):
        # write alignment with taxa removed into a file
        prefix = self.reduced_file_prefix
        alns = self.remove_taxa(self.species_to_remove)
        for file_name, aln_dict in alns.items():
            out_file_name = prefix + file_name + extension
            self.file_overwrite_error(out_file_name)
            self.write_formatted_file(file_format, out_file_name, aln_dict)
        return len(alns)

    def write_translated(self, index, alignment, file_format, extension):
        # write alignments translated into amino acids
        prefix = "translated_"
        file_name = self.get_alignment_name(index, extension)
        out_file_name = prefix + file_name + extension
        self.file_overwrite_error(out_file_name)
        self.write_formatted_file(file_format, out_file_name, alignment)

    def write_trimmed(self, index, alignment, file_format, extension):
        # write trimmed alignments
        if self.trim_out:
            out_file_name = self.trim_out
        else:
            prefix = "trimmed_"
            file_name = self.get_alignment_name(index, extension)
            out_file_name = prefix + file_name
        self.file_overwrite_error(out_file_name)
        self.write_formatted_file(file_format, out_file_name, alignment)

    def write_metapartitions(self, file_format):
        # write metapartitions - combines split and concat
        print("write_out elif action == metapartitions")
        metapartition_extension = self.get_metapartition_extension(file_format)
        list_of_alignments = self.get_partitioned(self.split)
        written_split_files = []
        err_indx = 0

        for item in list_of_alignments:
            try:
                for split_file in self.write_split(item, file_format, metapartition_extension):
                    written_split_files.append(split_file)
            except ValueError as e:
                    print("WARNING: ", e)
                    err_indx += 1
        if len(written_split_files) > 0:
            print("Wrote %d %s metapartition files from partitions provided" %
(len(written_split_files), file_format))
        if err_indx > 0:
            print("WARNING: %d file(s) raised an error while writing (see above)." % err_indx)

        # now set inputs to be the collated metapartition alignment files
        self.in_files = written_split_files
        self.alignment_objects = self.get_alignment_objects()
        self.parsed_alignments = self.get_parsed_alignments()

        # concat metapartition alignment files
        self.write_concat(file_format)

    def write_out(self, action, file_format):
        # write other output files depending on command (action)
        extension = self.get_extension(file_format)

        if action == "concat":
            self.write_concat(file_format)

        elif action == "convert":
            length = len(self.alignment_objects)
            [
                self.write_convert(i, alignment, file_format, extension)
                for i, alignment in enumerate(self.parsed_alignments)
            ]
            print("Converted " + str(length) + " files from " + self.in_format + " to " + file_format)

        elif action == "replicate":
            [
                self.write_replicate(i, alignment, file_format, extension)
                for i, alignment in enumerate(self.get_replicate(self.no_replicates, self.no_loci))
            ]

            print("Constructed " + str(self.no_replicates) + " replicate data sets, each from " +
str(self.no_loci) + " alignments")

        elif action == "split":
            list_of_alignments = self.get_partitioned(self.split)
            written_split_files = []
            err_indx = 0

            for item in list_of_alignments:
                try:
                    for split_file in self.write_split(item, file_format, extension):
                        written_split_files.append(split_file)
                except ValueError as e:
                        print("WARNING: ", e)
                        err_indx += 1
            if len(written_split_files) > 0:
                print("Wrote %d %s files from partitions provided" % (len(written_split_files), file_format))
            if err_indx > 0:
                print("WARNING: %d file(s) raised an error while writing (see above)." % err_indx)

        elif action == "metapartitions":
            self.write_metapartitions(file_format)

        elif action == "remove":
            aln_no = self.write_reduced(file_format, extension)
            if aln_no:
                print("Wrote " + str(aln_no) + " " + str(file_format) + " files with reduced taxon set")

        elif action == "translate":
            if self.data_type == "aa":
                print("ERROR: cannot translate; you said your alignment already contains amino acids")
                sys.exit()
            translated_alignment_dicts = self.get_translated(self.genetic_code, self.reading_frame)
            length = len(self.alignment_objects)
            [
                self.write_translated(i, alignment, file_format, extension)
                for i, alignment in enumerate(translated_alignment_dicts)
            ]
            print("Translated " + str(length) + " files to amino acid sequences")

        elif action == "trim": # self.trim_fraction, self.parsimony_check
            trimmed_alignment_dicts = self.get_trimmed(self.trim_fraction, self.parsimony_check)
            length = len(self.alignment_objects)
            [
                self.write_trimmed(i, alignment, file_format, extension)
                for i, alignment in enumerate(trimmed_alignment_dicts)
            ]
            print("Trimmed", str(length), "file(s) to have", self.trim_fraction, "minimum occupancy per alignment column")


def main():

    # initialize parsed arguments and meta alignment objects
    kwargs = run()
    meta_aln = MetaAlignment(**kwargs)

    if meta_aln.command == "summary":
        meta_aln.write_summaries(kwargs["summary_out"])
    if meta_aln.by_taxon_summary:
        print("Printing taxon summaries")
        meta_aln.write_taxa_summaries()
    if meta_aln.command == "convert":
        meta_aln.write_out("convert", kwargs["out_format"])
    if meta_aln.command == "concat":
        meta_aln.write_out("concat", kwargs["out_format"])
        meta_aln.write_partitions(kwargs["concat_part"], kwargs["part_format"], kwargs["data_type"], kwargs["codons"])
    if meta_aln.command == "replicate":
        meta_aln.write_out("replicate", kwargs["out_format"])
    if meta_aln.command == "split":
        meta_aln.write_out("split", kwargs["out_format"])
    if meta_aln.command == "remove":
        meta_aln.write_out("remove", kwargs["out_format"])
    if meta_aln.command == "translate":
        meta_aln.write_out("translate", kwargs["out_format"])
    if meta_aln.command == "trim":
        meta_aln.write_out("trim", kwargs["out_format"])

    if meta_aln.command == "metapartitions":
        # `metapartitions` is essentially `split` + `concat`. Currently you can't set an out_format:
        # it's automatically set to match the in_format because the intermediate `split` outputs become
        # the 'new' in_files for the `concat` operation, and then calling either:
        #  -> AminoAcidAlignment(Alignment.__init__(self, in_file, in_format, data_type))
        #  -> DNAAlignment(Alignment.__init__(self, in_file, in_format, data_type))
        # through MetaAlignment.get_alignment_object(alignment, self.in_format, self.data_type)
        meta_aln.write_out("metapartitions", kwargs["in_format"])
        meta_aln.write_partitions(kwargs["concat_part"], kwargs["part_format"], kwargs["data_type"], "none")

        # meta_aln.write_out("translate", kwargs["out_format"])

def run():

    # initialize parsed arguments
    config = ParsedArgs()
    # get arguments
    config_dict = config.get_args_dict()
    return config_dict

if __name__ == ’__main__’:
    main()
```

8.5 md/README.md File Reference

Index

__all__
    amas, 19
__author__
    amas, 19
__email__
    amas, 19
__enter__
    amas.AMAS.FileHandler, 54
__exit__
    amas.AMAS.FileHandler, 54
__init__
    amas.AMAS.Alignment, 25
    amas.AMAS.FileHandler, 54
    amas.AMAS.FileParser, 56
    amas.AMAS.MetaAlignment, 66
    amas.AMAS.ParsedArgs, 116
__str__
    amas.AMAS.Alignment, 25
__version__
    amas, 19

add_common_args
    amas.AMAS.ParsedArgs, 116
alignment_objects
    amas.AMAS.MetaAlignment, 107
all_matrix_cells
    amas.AMAS.Alignment, 42
all_same
    amas.AMAS.Alignment, 25
alphabet
    amas.AMAS.AminoAcidAlignment, 47
    amas.AMAS.DNAAlignment, 53
AMAS, 1
amas, 19
    __all__, 19
    __author__, 19
    __email__, 19
    __version__, 19
amas.AMAS, 20
    main, 20
    proportion, 21
    run, 22
amas.AMAS.Alignment, 23
    __init__, 25
    __str__, 25
    all_matrix_cells, 42
    all_same, 25
    append_count, 26
    check, 42
    check_data_type, 26
    data_type, 42
    get_alignment_length, 27
    get_aln_input, 27
    get_char_summary, 28
    get_column, 29
    get_counts, 29
    get_counts_from_parsed, 30
    get_counts_from_seq, 31
    get_matrix_cells, 31
    get_missing, 31
    get_missing_from_parsed, 31
    get_missing_from_seq, 32
    get_missing_percent, 33
    get_missing_percent_from_seq, 33
    get_name, 34
    get_parsed_aln, 34
    get_parsimony_informative, 35
    get_prop_parsimony, 35
    get_prop_variable, 35
    get_site_no_missing_ambiguous, 35
    get_sites_no_missing_ambiguous, 36
    get_taxa_no, 36
    get_taxon_char_summary, 37
    get_trim_selection, 38
    get_variable, 38
    in_file, 42
    in_format, 43
    length, 43
    matrix, 43
    matrix_creator, 39
    missing, 43
    missing_records, 43
    no_missing_ambiguous, 43
    parsed_aln, 44
    parsimony_informative, 44
    prop_parsimony, 44
    prop_variable, 44
    replace_missing, 39
    summarize_alignment, 40
    summarize_alignment_by_taxa, 41
    variable_sites, 44
amas.AMAS.AminoAcidAlignment, 45
    alphabet, 47
    get_summary, 46
    get_taxa_summary, 46
    missing_ambiguous_chars, 47
    missing_chars, 47
    non_alphabet, 47
amas.AMAS.DNAAlignment, 48

  alphabet, 53
  get_atgc_content, 49
  get_atgc_from_parsed, 49
  get_atgc_from_seq, 50
  get_list_from_atgc, 51
  get_summary, 51
  get_taxa_summary, 52
  missing_ambiguous_chars, 53
  missing_chars, 53
  non_alphabet, 53
amas.AMAS.FileHandler, 53
  __enter__, 54
  __exit__, 54
  __init__, 54
  file_name, 55
  get_file_name, 54
  in_file, 55
amas.AMAS.FileParser, 55
  __init__, 56
  chars_match, 61
  counter, 61
  fasta_parse, 56
  in_file, 61
  in_file_lines, 61
  matches, 62
  name_match, 62
  name_matches, 62
  nexus_interleaved_parse, 57
  nexus_parse, 58
  partitions_parse, 58
  phylip_interleaved_parse, 60
  phylip_parse, 60
  records, 62
  seq_match, 62
  seq_matches, 63
  sequence, 63
  sequences, 63
  tax_chars_matches, 63
  tax_match, 63
  taxa, 63
  translate_ambiguous, 60
amas.AMAS.MetaAlignment, 64
  __init__, 66
  alignment_objects, 107
  by_taxon_summary, 107
  check_align, 108
  check_taxa, 108
  codes, 108
  codes_list, 108
  codons, 108
  command, 108
  concat_out, 108
  cores, 109
  data_type, 109
  file_overwrite_error, 69
  gencode_NCBI_1, 109
  gencode_NCBI_10, 109
  gencode_NCBI_11, 109
  gencode_NCBI_12, 109
  gencode_NCBI_13, 110
  gencode_NCBI_14, 110
  gencode_NCBI_16, 110
  gencode_NCBI_2, 110
  gencode_NCBI_21, 110
  gencode_NCBI_22, 110
  gencode_NCBI_23, 110
  gencode_NCBI_24, 110
  gencode_NCBI_25, 111
  gencode_NCBI_26, 111
  gencode_NCBI_3, 111
  gencode_NCBI_4, 111
  gencode_NCBI_5, 111
  gencode_NCBI_6, 111
  gencode_NCBI_9, 111
  genetic_code, 111
  get_alignment_name, 70
  get_alignment_name_no_ext, 70
  get_alignment_object, 71
  get_alignment_objects, 71
  get_concatenated, 72
  get_extension, 73
  get_metapartition_extension, 74
  get_parsed_alignments, 74
  get_partitioned, 75
  get_partitions, 76
  get_replicate, 76
  get_summaries, 77
  get_taxon_summaries, 78
  get_translated, 79
  get_trimmed, 80
  in_files, 112
  in_format, 112
  natural_sort, 81
  no_loci, 112
  no_mpan, 112
  no_replicates, 112
  no_sup_aln_name, 112
  parsed_alignments, 113
  parsimony_check, 113
  prepend_label, 113
  print_fasta, 81
  print_iqtree_nexus_partitions, 82
  print_nexus, 83
  print_nexus_int, 84
  print_nexus_partitions, 85
  print_phylip, 86
  print_phylip_int, 87
  print_raxml_partitions, 87
  print_unspecified_partitions, 89
  reading_frame, 113
  reduced_file_prefix, 113
  remove_empty, 113
  remove_empty_sequences, 90
  remove_from_alignment, 90
  remove_taxa, 91
  remove_unknown_chars, 92

- replace_string_in_file, 92
- species_to_remove, 114
- species_to_remove_set, 114
- split, 114
- summarize_alignments, 93
- summarize_alignments_taxa, 93
- translate_dict, 94
- translate_dna_to_aa, 94
- trim_dict, 95
- trim_fraction, 114
- trim_out, 114
- using_metapartitions, 114
- write_concat, 95
- write_convert, 96
- write_formatted_file, 97
- write_metapartitions, 98
- write_out, 99
- write_partitions, 101
- write_reduced, 102
- write_replicate, 103
- write_split, 103
- write_summaries, 104
- write_taxa_summaries, 105
- write_translated, 105
- write_trimmed, 106

amas.AMAS.ParsedArgs, 115
- __init__, 116
- add_common_args, 116
- args, 126
- concat, 117
- convert, 118
- get_args_dict, 119
- metapartitions, 119
- remove, 121
- replicate, 121
- split, 122
- summary, 123
- translate, 124
- trim, 125

amas/__init__.py, 127
amas/AMAS.py, 127, 128
append_count
- amas.AMAS.Alignment, 26

args
- amas.AMAS.ParsedArgs, 126

by_taxon_summary
- amas.AMAS.MetaAlignment, 107

chars_match
- amas.AMAS.FileParser, 61

check
- amas.AMAS.Alignment, 42

check_align
- amas.AMAS.MetaAlignment, 108

check_data_type
- amas.AMAS.Alignment, 26

check_taxa
- amas.AMAS.MetaAlignment, 108

codes
- amas.AMAS.MetaAlignment, 108

codes_list
- amas.AMAS.MetaAlignment, 108

codons
- amas.AMAS.MetaAlignment, 108

command
- amas.AMAS.MetaAlignment, 108

concat
- amas.AMAS.ParsedArgs, 117

concat_out
- amas.AMAS.MetaAlignment, 108

convert
- amas.AMAS.ParsedArgs, 118

cores
- amas.AMAS.MetaAlignment, 109

counter
- amas.AMAS.FileParser, 61

data_type
- amas.AMAS.Alignment, 42
- amas.AMAS.MetaAlignment, 109

fasta_parse
- amas.AMAS.FileParser, 56

file_name
- amas.AMAS.FileHandler, 55

file_overwrite_error
- amas.AMAS.MetaAlignment, 69

gencode_NCBI_1
- amas.AMAS.MetaAlignment, 109

gencode_NCBI_10
- amas.AMAS.MetaAlignment, 109

gencode_NCBI_11
- amas.AMAS.MetaAlignment, 109

gencode_NCBI_12
- amas.AMAS.MetaAlignment, 109

gencode_NCBI_13
- amas.AMAS.MetaAlignment, 110

gencode_NCBI_14
- amas.AMAS.MetaAlignment, 110

gencode_NCBI_16
- amas.AMAS.MetaAlignment, 110

gencode_NCBI_2
- amas.AMAS.MetaAlignment, 110

gencode_NCBI_21
- amas.AMAS.MetaAlignment, 110

gencode_NCBI_22
- amas.AMAS.MetaAlignment, 110

gencode_NCBI_23
- amas.AMAS.MetaAlignment, 110

gencode_NCBI_24
- amas.AMAS.MetaAlignment, 110

gencode_NCBI_25
- amas.AMAS.MetaAlignment, 111

gencode_NCBI_26
- amas.AMAS.MetaAlignment, 111

gencode_NCBI_3

amas.AMAS.MetaAlignment, 111
gencode_NCBI_4
    amas.AMAS.MetaAlignment, 111
gencode_NCBI_5
    amas.AMAS.MetaAlignment, 111
gencode_NCBI_6
    amas.AMAS.MetaAlignment, 111
gencode_NCBI_9
    amas.AMAS.MetaAlignment, 111
genetic_code
    amas.AMAS.MetaAlignment, 111
get_alignment_length
    amas.AMAS.Alignment, 27
get_alignment_name
    amas.AMAS.MetaAlignment, 70
get_alignment_name_no_ext
    amas.AMAS.MetaAlignment, 70
get_alignment_object
    amas.AMAS.MetaAlignment, 71
get_alignment_objects
    amas.AMAS.MetaAlignment, 71
get_aln_input
    amas.AMAS.Alignment, 27
get_args_dict
    amas.AMAS.ParsedArgs, 119
get_atgc_content
    amas.AMAS.DNAAlignment, 49
get_atgc_from_parsed
    amas.AMAS.DNAAlignment, 49
get_atgc_from_seq
    amas.AMAS.DNAAlignment, 50
get_char_summary
    amas.AMAS.Alignment, 28
get_column
    amas.AMAS.Alignment, 29
get_concatenated
    amas.AMAS.MetaAlignment, 72
get_counts
    amas.AMAS.Alignment, 29
get_counts_from_parsed
    amas.AMAS.Alignment, 30
get_counts_from_seq
    amas.AMAS.Alignment, 31
get_extension
    amas.AMAS.MetaAlignment, 73
get_file_name
    amas.AMAS.FileHandler, 54
get_list_from_atgc
    amas.AMAS.DNAAlignment, 51
get_matrix_cells
    amas.AMAS.Alignment, 31
get_metapartition_extension
    amas.AMAS.MetaAlignment, 74
get_missing
    amas.AMAS.Alignment, 31
get_missing_from_parsed
    amas.AMAS.Alignment, 31
get_missing_from_seq
    amas.AMAS.Alignment, 32
get_missing_percent
    amas.AMAS.Alignment, 33
get_missing_percent_from_seq
    amas.AMAS.Alignment, 33
get_name
    amas.AMAS.Alignment, 34
get_parsed_alignments
    amas.AMAS.MetaAlignment, 74
get_parsed_aln
    amas.AMAS.Alignment, 34
get_parsimony_informative
    amas.AMAS.Alignment, 35
get_partitioned
    amas.AMAS.MetaAlignment, 75
get_partitions
    amas.AMAS.MetaAlignment, 76
get_prop_parsimony
    amas.AMAS.Alignment, 35
get_prop_variable
    amas.AMAS.Alignment, 35
get_replicate
    amas.AMAS.MetaAlignment, 76
get_site_no_missing_ambiguous
    amas.AMAS.Alignment, 35
get_sites_no_missing_ambiguous
    amas.AMAS.Alignment, 36
get_summaries
    amas.AMAS.MetaAlignment, 77
get_summary
    amas.AMAS.AminoAcidAlignment, 46
    amas.AMAS.DNAAlignment, 51
get_taxa_no
    amas.AMAS.Alignment, 36
get_taxa_summary
    amas.AMAS.AminoAcidAlignment, 46
    amas.AMAS.DNAAlignment, 52
get_taxon_char_summary
    amas.AMAS.Alignment, 37
get_taxon_summaries
    amas.AMAS.MetaAlignment, 78
get_translated
    amas.AMAS.MetaAlignment, 79
get_trim_selection
    amas.AMAS.Alignment, 38
get_trimmed
    amas.AMAS.MetaAlignment, 80
get_variable
    amas.AMAS.Alignment, 38

in_file
    amas.AMAS.Alignment, 42
    amas.AMAS.FileHandler, 55
    amas.AMAS.FileParser, 61
in_file_lines
    amas.AMAS.FileParser, 61
in_files
    amas.AMAS.MetaAlignment, 112
in_format

    amas.AMAS.Alignment, 43
    amas.AMAS.MetaAlignment, 112

length
    amas.AMAS.Alignment, 43

main
    amas.AMAS, 20
matches
    amas.AMAS.FileParser, 62
matrix
    amas.AMAS.Alignment, 43
matrix_creator
    amas.AMAS.Alignment, 39
md/README.md, 157
metapartitions
    amas.AMAS.ParsedArgs, 119
missing
    amas.AMAS.Alignment, 43
missing_ambiguous_chars
    amas.AMAS.AminoAcidAlignment, 47
    amas.AMAS.DNAAlignment, 53
missing_chars
    amas.AMAS.AminoAcidAlignment, 47
    amas.AMAS.DNAAlignment, 53
missing_records
    amas.AMAS.Alignment, 43

name_match
    amas.AMAS.FileParser, 62
name_matches
    amas.AMAS.FileParser, 62
natural_sort
    amas.AMAS.MetaAlignment, 81
nexus_interleaved_parse
    amas.AMAS.FileParser, 57
nexus_parse
    amas.AMAS.FileParser, 58
no_loci
    amas.AMAS.MetaAlignment, 112
no_missing_ambiguous
    amas.AMAS.Alignment, 43
no_mpan
    amas.AMAS.MetaAlignment, 112
no_replicates
    amas.AMAS.MetaAlignment, 112
no_sup_aln_name
    amas.AMAS.MetaAlignment, 112
non_alphabet
    amas.AMAS.AminoAcidAlignment, 47
    amas.AMAS.DNAAlignment, 53

parsed_alignments
    amas.AMAS.MetaAlignment, 113
parsed_aln
    amas.AMAS.Alignment, 44
parsimony_check
    amas.AMAS.MetaAlignment, 113
parsimony_informative
    amas.AMAS.Alignment, 44
partitions_parse
    amas.AMAS.FileParser, 58
phylip_interleaved_parse
    amas.AMAS.FileParser, 60
phylip_parse
    amas.AMAS.FileParser, 60
prepend_label
    amas.AMAS.MetaAlignment, 113
print_fasta
    amas.AMAS.MetaAlignment, 81
print_iqtree_nexus_partitions
    amas.AMAS.MetaAlignment, 82
print_nexus
    amas.AMAS.MetaAlignment, 83
print_nexus_int
    amas.AMAS.MetaAlignment, 84
print_nexus_partitions
    amas.AMAS.MetaAlignment, 85
print_phylip
    amas.AMAS.MetaAlignment, 86
print_phylip_int
    amas.AMAS.MetaAlignment, 87
print_raxml_partitions
    amas.AMAS.MetaAlignment, 87
print_unspecified_partitions
    amas.AMAS.MetaAlignment, 89
prop_parsimony
    amas.AMAS.Alignment, 44
prop_variable
    amas.AMAS.Alignment, 44
proportion
    amas.AMAS, 21

reading_frame
    amas.AMAS.MetaAlignment, 113
records
    amas.AMAS.FileParser, 62
reduced_file_prefix
    amas.AMAS.MetaAlignment, 113
remove
    amas.AMAS.ParsedArgs, 121
remove_empty
    amas.AMAS.MetaAlignment, 113
remove_empty_sequences
    amas.AMAS.MetaAlignment, 90
remove_from_alignment
    amas.AMAS.MetaAlignment, 90
remove_taxa
    amas.AMAS.MetaAlignment, 91
remove_unknown_chars
    amas.AMAS.MetaAlignment, 92
replace_missing
    amas.AMAS.Alignment, 39
replace_string_in_file
    amas.AMAS.MetaAlignment, 92
replicate
    amas.AMAS.ParsedArgs, 121
run

amas.AMAS, 22

seq_match
    amas.AMAS.FileParser, 62
seq_matches
    amas.AMAS.FileParser, 63
sequence
    amas.AMAS.FileParser, 63
sequences
    amas.AMAS.FileParser, 63
species_to_remove
    amas.AMAS.MetaAlignment, 114
species_to_remove_set
    amas.AMAS.MetaAlignment, 114
split
    amas.AMAS.MetaAlignment, 114
    amas.AMAS.ParsedArgs, 122
summarize_alignment
    amas.AMAS.Alignment, 40
summarize_alignment_by_taxa
    amas.AMAS.Alignment, 41
summarize_alignments
    amas.AMAS.MetaAlignment, 93
summarize_alignments_taxa
    amas.AMAS.MetaAlignment, 93
summary
    amas.AMAS.ParsedArgs, 123

tax_chars_matches
    amas.AMAS.FileParser, 63
tax_match
    amas.AMAS.FileParser, 63
taxa
    amas.AMAS.FileParser, 63
translate
    amas.AMAS.ParsedArgs, 124
translate_ambiguous
    amas.AMAS.FileParser, 60
translate_dict
    amas.AMAS.MetaAlignment, 94
translate_dna_to_aa
    amas.AMAS.MetaAlignment, 94
trim
    amas.AMAS.ParsedArgs, 125
trim_dict
    amas.AMAS.MetaAlignment, 95
trim_fraction
    amas.AMAS.MetaAlignment, 114
trim_out
    amas.AMAS.MetaAlignment, 114

using_metapartitions
    amas.AMAS.MetaAlignment, 114

variable_sites
    amas.AMAS.Alignment, 44

write_concat
    amas.AMAS.MetaAlignment, 95
write_convert
    amas.AMAS.MetaAlignment, 96
write_formatted_file
    amas.AMAS.MetaAlignment, 97
write_metapartitions
    amas.AMAS.MetaAlignment, 98
write_out
    amas.AMAS.MetaAlignment, 99
write_partitions
    amas.AMAS.MetaAlignment, 101
write_reduced
    amas.AMAS.MetaAlignment, 102
write_replicate
    amas.AMAS.MetaAlignment, 103
write_split
    amas.AMAS.MetaAlignment, 103
write_summaries
    amas.AMAS.MetaAlignment, 104
write_taxa_summaries
    amas.AMAS.MetaAlignment, 105
write_translated
    amas.AMAS.MetaAlignment, 105
write_trimmed
    amas.AMAS.MetaAlignment, 106