COLDP

Version

Table of Contents

coldp

Python tools for working with taxonomic checklists organised as Catalogue of Life Data Package (COLDP) format

Overview

coldp is a Python package to facilitate creation, manipulation, editing and serialisation of taxonomic checklists in the COL Data Package (COLDP) format.

The package includes three classes:

- **coldp.COLDP** - A COLDP package loaded as a set of Pandas dataframes
- **coldp.NameBundle** - A helper class to simplify addition of taxon names with sets of associated synonyms to a COLDP instance
- **coldp.IdentifierPolicy** - An internal class to manage ID values in COLDP dataframes

The main **COLDP** class instantiates a COLDP package in memory as a set of Pandas dataframes. An instance may be initialised from the contents of a folder containing a set of COLDP-compliant CSV or tab-delimited data files or alternatively can be initialised as an empty instance in memory. The class includes many methods for inserting new data, editing existing records and querying the contents of the package. The instance can then be saved as a set of CSV files in a named folder .

The **NameBundle** class brings together a scientific name and its synonyms so that these can be added together and the COLDP package can automatically manage their relationships NameBundle objects are normally created using the `coldp.COLDP.start_name_bundle()` method.

At minimum a NameBundle is initialised with a dictionary holding a set of COLDP name record values. The scientific name represented by this dictionary should be the accepted name for a species or other taxon. Synonyms may then be added to the NameBundle.

Once all names are included, the NameBundle can be added to the COLDP object via the `coldp.COLDP.add_names()` method. This adds name, taxon and synonym records for the set of names supplied. COLDP options may expand the set of added synonyms to include variant formats or may trigger the addition of one or taxa that are implicit in the accepted name.

The COLDP object will automatically manage record identifiers and the basionymID for any name records that are combinations of another name in the set.

Installation

Install the latest version from PyPI.

```
pip install py-coldp
```

Classes

coldp.COLDP

Class

COLDP. __init__ (*name* , *folder = '.'* , *code = 'ICZN'* , *default_taxon_record = None* , *insert_species_for_trinomials = False* , *create_subspecies_for_infrasubspecifics = False* , *create_synonyms_without_subgenus = False* , *basionyms_from_synonyms = False* , *classification_from_parents = False* , *allow_repeated_binomials = False* , *create_taxa_for_not_established = False* , *issues_to_stdout = False* , *context = None*)

Class to manage a set of Pandas dataframes for CSV tables in Catalogue of Life Data Package.

Parameters :

- **name** – The name for the COLDP package. If a folder of this name exists inside the folder specied by the folder parameter, this COLDP instance will be initialised with the contents of any COLDP-compliant CSV or TSV files in the named folder.
- **folder** – Name of folder that may contain the named COLDP source folder from which source files should be read (name specified via the name parameter) and that is the default folder in which a COLDP folder will be written when the COLDP instance is saved. This is not the folder in which the CSV files are written. It is the folder which will contain the subfolder holding CSV files.
- **code** – ICZN or ICBN to indicate the nomenclatural code for name formatting. ICZN is the default.
- **default_taxon_record** – Any values in the dictionary are automatically used as default values in taxon records added to the COLDP instance. In other words, the dictionary becomes the base into which other taxon record properties are then inserted.

- **insert_species_for_trinomials** – If true, insert species (taxon and name) if trinomials are provided without the associated species. This can be convenient when mapping source data that only lists infraspecific taxa for species with subspecies, varieties or forms.

- **create_subspecies_for_infrasubspecifics** – If true, automatically add a trinomial at subspecific rank as a synonym whenever an infrasubspecific name is added. This may be useful if the resulting dataset is intended to provide easy mapping of strings to taxon concepts, since versions of the trinomial lacking the rank marker will often be found in source data.

- **create_synonyms_without_subgenus** – If true, automatically add a binomial or trinomial without an included subgenus name as a synonym whenever a name including a subgenus is added. This may be useful if the resulting dataset is intended to provide easy mapping of strings to taxon concepts, since versions of lacking the subgeneric component will often be found in source data.

- **basionyms_from_synonyms** – If true, basionym associations are automatically created from synonyms within a loaded COLDP dataset. If an accepted name includes parentheses around the authorship, and if a name with the same epithet and authorship but no parentheses is included in the synonyms, the ID value for the synonym will be used for the basionymID element in the accepted name. This is normally a housekeeping step when first loading a new COLDP dataset. This option is only used when the dataset is first loaded. Addition of basionym relationships is automatic for names added as part of the same NameBundle.

- **classification_from_parents** – If true, insert names for higher ranks into relevant elements in each taxon record based on the parent and higher ancestry for the taxon.

- **allow_repeated_binomials** – If true, omit checks rejecting the addition of the same binomial more than once to the COLDP name dataframe.

- **create_taxa_for_not_established** – If true, generate taxon records even if the associated name is flagged as not

established (i.e. not satisfying the relevant nomenclatural code)

- **issues_to_stdout** – If true, print any issue messages (see `issue()`) to stdout as well as inserting then in the issue dataframe.
- **context** – Value to be used in labeling issue records (see `issue()`). This value is more normally set using `set_context()` .

A COLDP object is initialised with a source/destination folder, COLDP package name and other options.

If a subfolder exists with the supplied name in the supplied folder, it will be initialised with data from any CSV/TSV files it contains with any of the following base names: name, taxon, synonym, reference, distribution, speciesinteraction, namerelation, typematerial or nameusage, and with a file extension recognised via `csv_extensions` . File names are case insensitive.

Files with the base name nameusage are loaded only if the name, taxon or synonym file is absent, in which case the contents of the nameusage file are mapped internally to the more normalised COLDP format with separate name + taxon + synonym tables.

If no folder exists with the supplied name, an empty instance is created. Pandas dataframes are created as required for the COLDP data types as data are inserted into the instance.

The `start_name_bundle()` method produces a `NameBundle` object for preparing an accepted name and associated synonyms to pass to the `add_names()` method which creates name, taxon and synonym records as a set of cross-referenced objects.

Other data is added using the `add_names()` , `add_name_relation()` , `add_typematerial()` , `add_distribution()` , `add_species_interaction()` and `modify_taxon()` methods.

The `save()` method writes the data back to the same or another folder.

Methods

Control behaviour

COLDP. set_options (*** options*)

Set options controlling COLDP instance from keyword arguments

Parameters :

options – Set of boolean options for controlling the behaviour of the COLDP package - see `COLDP` for boolean options that can be set.

Convenience method for setting options via keyword arguments after the COLDP instance has been created.

COLDP. set_default_taxon_record (*default_taxon_record*)

Set default values for properties in taxon records created by this COLDP instance

Parameters :

default_taxon_record – Any values in the dictionary are automatically used as default values in taxon records added to the COLDP instance. In other words, the dictionary becomes the base into which other taxon record properties are then inserted.

Convenience method for setting default taxon record after the COLDP instance has been created.

COLDP. set_context (*context*)

Set a string label for annotating issues in the COLDP issue table

Parameters :

context – String to label any issues associated with data changes from the current point forward

The COLDP class automatically logs errors and issues to an issues dataframe. This dataframe is included among the CSV output files when a COLDP instance is saved.

Each row in the issue table indicates a possible problem with data added to the instance. The context variable provides an external mechanism for the user to label these issues as they occur. For example, if the user is processing a spreadsheet of species names, they can call set_context with a string identifying the row from the source spreadsheet. This value will be included in the context column in the issue.csv output.

Add or modify records

COLDP. start_name_bundle (*accepted* , *incertae_sedis = False* , *sic = False*)

Return a NameBundle object based on the supplied accepted name and referencing this COLDP instance

Parameters :

- **accepted** – Dictionary containing COLDP Name record for accepted name
- **incertae_sedis** – If True, the associated COLDP Taxon Record will be flagged as *incertae sedis*
- **sic** – If True, issues will not be reported if the name is not properly code-compliant

Returns :

NameBundle instance initialised with supplied parameters

This is the normal way to construct a new NameBundle object.

COLDP. add_names (*bundle* , *parent = None*)

Ensure one or more names are included in names dataframe and update taxa and synonyms dataframes as necessary

Parameters :

- **bundle** – `NameBundle` object with set of associated taxon names
- **parent** – ID for taxon record for which the accepted taxon in this bundle is to be added as a child

The supplied `NameBundle` object includes an accepted name record (as a dictionary of COLDP name properties) and a list, which may be empty, of synonym name records in the same format. `add_names()` ensures that name records exist for all these names in the names dataframe, that a taxon record exists for the accepted taxon name, and that synonym records exist for this taxon record for all supplied synonyms.

Depending on the options supplied for the `COLDP` object, additional synonyms may be created to represent subspecies-rank versions of infrasubspecific trinomials and subgenus-free versions of combinations including a subgenus name.

If a matching name already exists in the COLDP instance, no new name will normally be added. Instead, the name record in the bundle will be updated with the ID (and basionymID where applicable) from the existing name. This allows for additional synonyms to be added to an existing taxon. The behaviour can be over-ridden with the `allow_repeated_binomials` option, in which case any number of matching names can be added.

If the `insert_species_for_trinomials` option is set, a new species will be created if required before adding the trinomial as its child. In this case the bundle will contain the id for the species taxon as a species_taxon_id property.

Name records in the bundle are all updated with existing or new IDs and basionymIDs, which are inferred automatically for zoological names by associating combinations with and without parentheses around the authorship.

Upon completion, the bundle also contains an accepted_taxon_id property which includes the string ID for the taxon.

If `parent` is None, the new taxon record is created without a parent, i.e. becomes a root node in the classification.

COLDP. add_references (*reference_list*)

Ensure one or more references are included in references dataframe

Parameters :

reference_list – List of dictionaries - each dictionary contains values keyed by terms from reference_headings

Returns :

Updated `reference_list` with IDs for references in this COLDP instance

Find existing ID values for each supplied reference, based on identity of: author, title, issued, containerTitle, volume, issue, page and citation. Add ID from the appropriate reference dictionary in references. If none found, set ID to next unused index and add to references.

The list is returned updated with current ID values so these can be used for referenceID values in other classes.

COLDP. add_type_material (*type_material*)

Add COLDP TypeMaterial record to COLDP instance

Parameters :

type_material – COLDP TypeMaterial record represented as dictionary of properties

Returns :

TypeMaterial record returned unchanged

The nameID value must match the ID value for an existing name record.

This method returns the record unchanged (for consistency with other add records which may alter the provided record).

COLDP. add_distribution (*distribution*)

Add COLDP Distribution record to COLDP instance

Parameters :

distribution – COLDP Distribution record represented as dictionary of properties

Returns :

Distribution record returned unchanged

The taxonID value must match the ID value for an existing taxon record.

This method returns the record unchanged (for consistency with other add records which may alter the provided record).

COLDP. add_species_interaction (*interaction*)

Add COLDP SpeciesInteraction record to COLDP instance

Parameters :

interaction – COLDP SpeciesInteraction record represented as dictionary of properties

Returns :

SpeciesInteraction record returned unchanged

The taxonID value must match the ID value for an existing taxon record.

This method returns the record unchanged (for consistency with other add records which may alter the provided record).

COLDP. add_name_relation (*name_relation*)

Add COLDP NameRelation record to COLDP instance

Parameters :

name_relation – COLDP NameRelation record represented as dictionary of properties

Returns :

NameRelation record returned unchanged

The nameID and relatedNameID values must match the ID values for existing name records.

This method returns the record unchanged (for consistency with other add records which may alter the provided record).

COLDP. add_synonym (*synonym*)

Add COLDP Synonym record to COLDP instance

Parameters :

synonym – COLDP Synonym record represented as dictionary of properties

Returns :

Synonym record returned unchanged

The taxonID value must match the ID value for an existing taxon record and the nameID value must match the ID value for an existing name record.

This method returns the record unchanged (for consistency with other add records which may alter the provided record).

COLDP. modify_name (*name_id* , *properties*)

Add or modify properties on a COLDP Name record

Parameters :

- **taxon_id** – String ID for a Name record
- **properties** – Dictionary of COLDP Name properties to be set for the identified record

If a name record exists with the given ID, set all properties in the dictionary.

COLDP. modify_taxon (*taxon_id* , *properties*)

Add or modify properties on a COLDP Taxon record

Parameters :

- **taxon_id** – String ID for a Taxon record
- **properties** – Dictionary of COLDP Taxon properties to be set for the identified record

If a taxon record exists with the given ID, set all properties in the dictionary.

Save

COLDP. save (*destination = None* , *name = None*)

Write dataframes as COLDP CSV files

Parameters :

- **destination** – Path to folder in which package should be saved. Defaults to destination supplied to constructor, which defaults to "."
- **name** – Name for COLDP package (subfolder name). Defaults to name supplied to constructor, which defaults to None. If no name provided, save will fail.

If necessary creates subfolder with name `name` in `destination` , and then writes CSV file representations for all DataFrames in the folder. Empty columns are dropped. Any numpy NAN elements are replaced with an empty string.

Find or get records

COLDP. find_taxon (*scientificName* , *authorship* , *rank*)

Get COLDP Taxon record (as Pandas dataframe) with accepted name matching supplied scientificName, authorship and rank values

Parameters :

- **scientificName** – Scientific name in canonical format
- **authorship** – Authorship string
- **rank** – Rank name string

Returns :

COLDP Taxon record matching supplied parameters

Logs a warning issue if multiple taxon records exist for a matching name and returns the first such match. Returns None if no match.

COLDP. find_name_record (*name*)

Return record from names DataFrame matching key fields (scientificName, authorship and rank) in supplied record

Parameters :

name – Dictionary containing Name properties

Returns :

Dictionary containing matching record if found

Locates any existing record in the names DataFrame that matches the supplied scientificName, authorship and rank and returns it as a COLDP Name properties dictionary, or None if no match is found.

COLDP. find_names (*properties* , *to_dict = False*)

Get all COLDP Name records matching all the supplied properties

Parameters :

- **properties** – Dictionary of property values to serve as filter
- **to_dict** – True if records should be converted from Pandas format to dictionary records, False (default) otherwise.

Returns :

Set of COLDP Name records either as DataFrame or list of dictionaries

Returns all matching records as Pandas DataFrame or list of dictionaries. If no matches, None is returned.

COLDP. find_name (*scientificName* , *authorship* , *rank*)

Return record from names DataFrame matching supplied scientificName, authorship and rank

Parameters :

- **scientificName** – Scientific name in canonical format
- **authorship** – Authorship string
- **rank** – Rank name string

Returns :

Dictionary containing matching record if found

Locates any existing record in the names DataFrame that matches the supplied scientificName, authorship and rank and returns it as a COLDP Name properties dictionary, or None if no match is found.

If `authorship` is None, returns a name based only on scientificName and rank.

COLDP. find_reference (*reference*)

Locate existing COLDP reference record exactly matching all major fields in `reference`

Parameters :

reference – Dictionary of COLDP reference properties representing a record to be found

Returns :

DataFrame with one COLDP reference record if found, else None

Only returns a record that exactly matches the values supplied in `reference` for all of author, title, issued, containerTitle, volume, issue, page and citation.

COLDP. find_distribution (*distribution*)

Locate existing COLDP distribution record exactly matching all major fields in `distribution`

Parameters :

distribution – Dictionary of COLDP distribution properties representing a record to be found

Returns :

DataFrame with one COLDP distribution record if found, else None

Only returns a record that exactly matches the values supplied in `distribution` for all of taxonID, area, gazetteer, status, referenceID.

COLDP. find_species_interaction (*interaction*)

Locate existing COLDP speciesinteraction record exactly matching all major fields in `interaction`

Parameters :

interaction – Dictionary of COLDP speciesinteraction properties representing a record to be found

Returns :

DataFrame with one COLDP speciesinteraction record if found, else None

Only returns a record that exactly matches the values supplied in `interaction` for all of taxonID, relatedTaxonID, relatedTaxonScientificName, type, and referenceID.

COLDP. find_type_material (*type_material*)

Locate existing COLDP typematerial record exactly matching all major fields in `type_material`

Parameters :

type_material – Dictionary of COLDP typematerial properties representing a record to be found

Returns :

DataFrame with one COLDP typematerial record if found, else None

Only returns a record that exactly matches the values supplied in `type_material` for all of nameID, citation, status, locality, country, latitude, longitude, elevation, date, collector, institutionCode, sex and referenceID, .

COLDP. get_name (*id*)

Return record from names DataFrame with supplied ID

Parameters :

id – String ID for COLDP Name record

Returns :

Pandas DataFrame containing matching record if found

Locates any existing record in the names DataFrame with the supplied ID, or None if no match is found. If multiple matches are found, logs an issue and returns the first.

COLDP. get_reference (*id*)

Get reference record as dictionary from ID string

Parameters :

id – ID string for requested COLDP reference record

Returns :

Dictionary of COLDP reference properties for requested ID

Returns None if no matching reference. If multiple records exist with the given id, a warning is logged and the first match is returned.

COLDP. get_taxon (*id*)

Return a COLDP Taxon record matching the supplied ID

Parameters :

id – String ID value

Returns :

Dictionary representation of COLDP Taxon record

Logs a warning if more than one match and returns the first such match.

COLDP. get_synonyms (*taxonID* , *to_dict = False*)

Get all COLDP Synonym records for the supplied taxon ID

Parameters :

- taxonID – String taxonID value

- **to_dict** – True if records should be converted from Pandas format to dictionary records, False (default) otherwise.

Returns :

Set of COLDP Synonym records for taxon either as DataFrame or list of dictionaries

Returns all matching records as Pandas DataFrame or list of dictionaries. If no matches, None is returned.

COLDP. get_synonymy (*nameID* , *to_dict = False*)

Get accepted COLDP Name record and all synonym COLDP Name records for the supplied name ID

Parameters :

- **taxonID** – String nameID value
- **to_dict** – True if records should be converted from Pandas format to dictionary records, False (default) otherwise

Returns :

Tuple containing COLDP Name record for accepted name and a DataFrame or list of dictionaries representing all synonym Name records

Maps the name indicated by the provided nameID to the accepted taxon and returns its name and the names for all synonyms for the taxon. These may all be returned either as Pandas DataFrames or in dictionary representations.

COLDP. get_children (*taxonID* , *to_dict = False*)

Get all child taxa for the COLDP Taxon associated with the supplied taxonID

Parameters :

- **taxonID** – String ID for COLDP Taxon record
- **to_dict** – True if records should be converted from Pandas format to dictionary records, False (default) otherwise.

Returns :

Set of COLDP Taxon records for children of identified taxon
either as DataFrame or list of dictionaries

Returns all matching records as Pandas DataFrame or list of dictionaries. If no matches, None is returned.

Tidy package

COLDP. fix_basionyms (*names* , *synonyms*)

Update dataframe of name records so that subsequent combination names refer to the original basionym record where present

Parameters :

- **names** – Dataframe containing COLDP name records to be updated
- **synonyms** – Dataframe containing COLDP synonym records corresponding to the name records in names

For any name record in `names` if the name is a subsequent zoological combination (with parentheses around authorship), find any other record in `names` that matches the authorship, year and epithet but lacks parentheses and update the basionymID property of the name to refer to this second record's ID. `synonyms` is used to assist with selection of the correct match in cases where more than one possible record is found.

The parameters are the two relevant DataFrames from the same COLDP instance.

COLDP. fix_classification ( )

Tidy and fill in higher classification for taxon records based on hierarchy

Recursively fixes classification elements for whole taxa dataframe

COLDP. sort_taxa ( )

Sort taxa dataframe so that all taxon records are sequenced hierarchically and alphabetically.

Following this method, the taxon table is sorted so that any taxa without parents are sorted alphabetically by scientific nameand each is followed immediately by its children and their descendents also sorted alphabetically. The result is a tree with siblings ordered alphabetically.

Existing ID values for all records are unchanged.

This is a convenience method to simplify review of the data in a spreadsheet or editor tool. It also simplifies import of the data into a database since there are no forward references to parent taxa.

COLDP. sort_names ( )

Sort name records to match order of records in taxa dataframe and with accepted names and synonyms sorted alphabetically

Records are sorted first in the current order of the taxon table and then alphabetically within the set of names for each taxon.

Existing ID values for all records are unchanged.

This method is most useful following a call to `sort_taxa()` .

COLDP. reset_ids (*name = None* , *prefix = None*)

Reset all IDs for one or more COLDP data tables to consecutive values in the order of the table records

Parameters :

- **name** – Table to be modified. One of name, taxon, reference or synonym. If name is None, all four are processed
- **prefix** – Optional prefix string to be prepended before consecutive integer record ids

Resets all ids for one or all of the four supported classes to consecutive id values, with an optional string prefix which defaults to `"s_"` in the case of synonym records and is otherwise empty.

Also modifies corresponding foreign ID references in other tables so they continue to resolve correctly using the `id_mappings` dictionary.

If no table name is specified, all four are processed.

Utilities

COLDP. get_text_tree (*taxonID* , *indent = ' '* , *initial_prefix = ''* , *synonym_prefix = '='*)

Generate formatted text tree for specified taxon and its descendents

Parameters :

- **taxonID** – String ID for taxon to be formatted

- **indent** – Indent string to be added one or more times before each nested row, defaults to two spaces
- **initial_prefix** – Optional prefix string for all rows (preceeds indentation), defaults to empty string
- **synonym_prefix** – Prefix to appear before synonymised names, defaults to an equal sign with spaces on either side

Returns :

Multiline string representation of taxonomic hierarchy

The tree view shows the name, authorship and rank for the selected taxon. Synonyms follow, one to a row at the same indent level but preceded by a space and an asterisk. Then each child follows, indented two more spaces per level, with its own synonyms and children following.

COLDP. **get_available_column_headings** ( )

Return dictionary mapping table names to lists of supported columns

Returns :

Dictionary containing copies of internal heading lists

Returns copies to avoid corrupting the lists used in this class

COLDP. **get_identifier_policy** (*table_name* , *default_prefix = None* , *required = True* , *volatile = False*)

Find or create `IdentifierPolicy` associated with named table

Parameters :

- **table_name** – Name of COLDP dataframe for which policy is required
- **default_prefix** – String prefix to use before numeric ID values if existing ID values are not consistently positive integer values
- **required** – Flag to indicate whether the table must have ID values - if False, the `IdentifierPolicy` will return None unless `existing` already contains ID values

- **volatile** – Flag to indicate if external code may modify ID values while the current COLDP instance is active - if False, the policy and future values will be determined on initialisation, otherwise the policy will be reviewed for each new ID value

Returns :

`IdentifierPolicy` object or None if no policy is required for the table

Creates new `IdentifierPolicy` instance if this is the first invocation for the given table. Policies take into account any existing ID values for the table.

Access to DataFrames

COLDP. table_by_name (*name*)

Return dataframe for named COLDP data class

Parameters :

name – Name of data frame to return (one of: name, taxon, synonym, reference, type_material, distribution, species_interaction, name_relation)

Returns :

Requested dataframe or None if no such table exists

Returns dataframe if one exists for supplied name.

Constants

coldp. csv_extensions

Dictionary mapping supported CSV/TSV file extensions to the expected delimiter.

Supported extensions are .csv for comma-separated data files and .tsv or .txt for tab-delimited data files.

coldp. id_mappings

Dictionary mapping table names (name, taxon, reference or synonym) to a dictionary that maps another table name to the properties in the second table that reference ID values from the first table.

For example, `id_mappings` maps the key “name” to a dictionary that includes the key “namerelation” with a list containing “nameID” and “relatedNameID” as its value.

This is a map of the foreign-key relationships that need to be validated and preserved between the COLDP data tables.

coldp. name_from_nameusage
Dictionary mapping names of columns in COLDP Name records to the corresponding column names in COLDP NameUsage records

coldp. taxon_from_nameusage
Dictionary mapping names of columns in COLDP Taxon records to the corresponding column names in COLDP NameUsage records

coldp. synonym_from_nameusage
Dictionary mapping names of columns in COLDP Synonym records to the corresponding column names in COLDP NameUsage records

Internal methods

COLDP. initialise_dataframe (*foldername* , *name* , *default_headings*)
Load or create a dataframe for one of the COLDP record types

Parameters :

- foldername – Path string for COLDP folder potentially containing serialised dataframe
- name – Base name for COLDP class (name, taxon, etc.) for which the dataframe is requested
- default_headings – Column headings to use if returning a new empty dataframe

Returns :

Dataframe for requested COLDP data class

Checks for a file in the COLDP folder with a supported extension (.csv, .tsv, .txt) and a name matching one of the COLDP record types (name, taxon, synonym, reference, distribution, typematerial or speciesinteraction). If this exists, it is loaded as a Pandas dataframe.

If it is not found but the name is one of name, taxon or synonym and a file with the name nameusage does exist, the relevant columns will be loaded from the nameusage file.

If no matching file is found, returns an empty dataframe.

COLDP. extract_table (*df* , *headings* , *mappings*)

Extract a set of columns from a dataframe using a dictionary that maps source columns names to target column names

Parameters :

- **df** – Dataframe from which columns are to be extracted/ mapped
- **headings** – List of column headings for destination dataframe
- **mappings** – Dictionary mapping destination column names to source column names

Returns :

New Dataframe based on supplied mappings

This method is used to extract and rename a subset of columns from a COLDP NameUsage table.

COLDP. insert_taxon (*name* , *parentID* , *incertae_sedis = False*)

Insert COLDP Taxon record as child of identified parent - creates or moves record as necessary

Parameters :

- **name** – COLDP Name record to be used as accepted name for new taxon
- **parentID** – String identifier for parent taxon

Returns :

Dictionary containing taxon record

If a taxon record already exists for the name, any parenthood change is made as required and the record is returned as a dictionary. Otherwise a new record is created and returned. Default values are taken from the `default_taxon_record` dictionary.

COLDP. insert_synonym (*taxon_id* , *name_id*)

Add basic COLDP Synonym record to COLDP instance

Parameters :

- **taxon_id** – String ID for taxon for which synonym is being registered
- **name_id** – String ID for name which is being registered as a synonym

Ensures that a synonym record exists in the COLDP instance for the given taxon and name.

COLDP. fix_classification_recursive (*taxa* , *ranks* , *parent = None*)

Recursively complete classification elements (higher taxon IDs) in taxon records descended from a given parent or for all taxon records in the COLDP instance at the highest included rank

Parameters :

- **taxa** – Dataframe containing COLDP taxon records
- **ranks** – DataFrame containing at least nameID, rank and scientificName for all names in COLDP instance (can be the complete table of COLDP names)
- **parent** – Taxon record for which children should be updated, or None if all top-level taxa should be updated

If `taxa` is None, select all top-level taxa from the dataframe (i.e. all without a parentID) and process these recursively.

If a parent is supplied, copy classification properties (kingdom, phylum, subphylum, class, subclass, order, suborder, superfamily, family, subfamily, tribe, subtribe, genus, subgenus, section, species) from the parent record, set any rank-specific column matching the rank of the parent to the name of the parent taxon, and then copy these rank values into all taxon records that are immediate descendents of the parent. This will ensure that the higher classification for all children matches this parent. Then call this method recursively for all children.

COLDP. sort_taxa_recursive (*df* , *ids* , *id*)

Internal method for recursive sorting of taxon records by parent and name

Parameters :

- **df** – Taxa dataframe to be sorted

- **ids** – Alternative list of IDs and preferred positions (as strings)
- **id** – Taxon ID for current taxon

Returns :

`ids` with additional values for current taxon and its descendents

Appends next taxon ID to the list along with a value guaranteed to be higher than the one added on the previous execution of this method. Recursively add IDs for children.

COLDP. prepare_bundle (*bundle*)

Add extra names to `NameBundle` if required based on current options

Parameters :

bundle – NameBundle to be processed

If `insert_species_for_trinomials` is True, ensure that the implied binomial exists for any new trinomials.

If `create_subspecies_for_infrasubspecifics` is True, add a subspecies-rank synonym to the bundle for each infrasubspecific name.

If `create_synonyms_without_subgenus` is True, add a subgenus-free synonym to the bundle for each name containing a subgenus.

COLDP. validate_record (*record_type* , *record*)

Verify whether all ID values used as forign keys in `record` correspond with existing records in the relevant tables

Parameters :

- **record_type** – Name for the data class to which this record should below
- **record** – Dictionary of COLDP properties for the record

Uses the `id_mappings` dictionary to identify which properties should be foreign keys. If these are present in the current record, check that the supplied value is indeed an ID from the relevant table.

COLDP. identify_name (*name*)

Ensure COLDP Name record is present and return a copy containing the current ID and basionymID

Parameters :

name – Dictionary containing COLDP Name properties

Returns :

Currently stored version of the name record in question

If a COLDP Name record for this name already exists (based on `find_name_record()`), return the existing name, unless this is a species name and the `allow_repeated_binomials` option has been set, in which case a new record will be created.

If the name is missing, create a new record with the next unused ID value and return the record with the ID.

COLDP. same_basionym (*a* , *b*)

Validates whether two name records share the same basionym (i.e. are different combinations for the same original name) :param _sphinx_paramlinks_coldp.COLDP.same_basionym.a: Dictionary with parameters representing a COLDP Name record :param _sphinx_paramlinks_coldp.COLDP.same_basionym.b: Dictionary with parameters representing a COLDP Name record :return: True if the parenthensis-free authorship and lowest-ranked epithets match, False otherwise.

COLDP. remove_gender (*epithet*)

Return epithet stripped on all likely gender-specific endings

Parameters :

epithet – String zoological epithet

Returns :

Epithet stripped of any possible Greek or Latin gender endings

Removes “a”, “us”, “um”, “is”, “e”, “os” or “on” as an ending.

COLDP. get_original_authorship (*authorship*)

Return authorship string without enclosing parentheses

Parameters :

authorship – Authorship string

Returns :

Authorship stripped of enclosing parentheses

Relevant only for zoological names

COLDP. epithet_and_authorship_match (*name* , *epithet* , *authorship*)

Check whether a name record matches the supplied epithet and authorship

Parameters :

- **name** – Dictionary containing COLDP Name record
- **epithet** – Specific or infraspecific epithet
- **authorship** – Authorship string

Returns :

True if the `epithet` matches the lowest-ranked epithet and `authorship` matches the authorship in the name record

Comparison ignores epithet gender endings.

COLDP. set_basionymid (*name* , *basionymid*)

Set basionymID on Name record in names DataFrame

Parameters :

- **name** – Name record as dictionary of COLDP properties
- **basionymid** – String ID of associated basionyn record

COLDP. fix_basionymid (*name* , *synonyms*)

Update name so its basionymID references the original combination in a list of synonyms

Parameters :

- **name** – Dictionary containing COLDP Name record for which basionymID is to be fixed
- **synonyms** – List of COLDP Name records that may contain basionym

Returns :

Returns name following any updates.

COLDP. construct_species_rank_name (*g* , *sg* , *s* , *ss* , *marker*)

Consistently construct a species or infraspecific name from the included epithets and optional rank marker

Parameters :

- **g** – Genus name
- **sg** – Subgenus name
- **s** – Specific epithet
- **ss** – Infraspecific epithet
- **marker** – Rank marker (e.g. “var.”) or rank name (“variety”)

Returns :

Complete scientific name string (no italics or authorship)

Convenience method for composing a scientific name with option subgenus, infraspecific epithet and rank marker. A variety of markers are handled and mapped to one of “var.”, “subvar.”, “f.” and “ab.”.

COLDP. construct_authorship (*a* , *y* , *is_basionym*)

Consistently construct a scientific name authorship from the included author names and year with parentheses where required

Parameters :

- **a** – Author string
- **y** – Publication year as string

- **is_basionym** – Boolean to indicate whether this is the original combination (basionym) or a subsequent combination that requires parentheses

Returns :

Tuple including 1) formatted authorship string and 2) publication year as a separate string

Convenience method for composing an authorship string. Includes parentheses if `is_basionym` is True.

If the `year` includes more than one part (e.g. “1832 [1831-1838]”), the first part (“1832”) is used for the year in the authorship string and the second part (“[1831-1838]”) is returned as the publication year. Otherwise, the same string is used in both cases.

COLDP. **is_species_group** (*name*)

Check whether name is in the species group (species or lower rank)

Parameters :

name – Dictionary representing a COLDP Name record

Returns :

True if the name is in the species group, False otherwise

Simply checks if the name includes a specificEpithet value. Any name passed to this method should include atomic name components and not just a scientificName.

COLDP. **is_infrasubspecific** (*name*)

Check whether name is in for a rank below the subspecies

Parameters :

name – Dictionary representing a COLDP Name record

Returns :

True if the name is infraspecific

Simply checks if the supplied rank is one of those below the subspecies.

COLDP. **issue** (*message*)

Log issue with data provided through methods

Parameters :

message – Text to be saved as body of issue

Creates a new record in an issues DataFrame that will be included in the COLDP export when `save()` is called. This record includes the message and the context string supplied on the most recent call to `set_context()` .

If `issues_to_stdout` is True, the context and message are also output as an error via logging.

Index and search

- Index
- Search Page

coldp.NameBundle

Class

NameBundle. __init__ (*coldp* , *accepted* , *incertae_sedis = False* , *sic = False*)

Wrapper class to manage set of associated names, normally an accepted name and one or more synonyms. The bundle handles the logic of associated name variations.

Parameters :

- **coldp** – `COLDP` object to handle logging of any issues and normalise name records
- **accepted** – Dictionary mapping COLDP name elements to values for the accepted name - a name record and a taxon record will be added to the COLDP package for the accepted name
- **incertae_sedis** – Flag to indicate if the resulting taxon record should be marked “incertae sedis”
- **sic** – Flag to indicate if the accepted name should be processed without reporting issues for invalid format

The minimal usage is to create a new bundle with an accepted name. One or more synonyms can also be supplied using the `add_synonym()` method. The bundle is then submitted to the `coldp.COLDP.add_names()` method for processing.

NameBundle objects should not be created directly. Use the `coldp.COLDP.start_name_bundle()` method instead.

accepted should contain a dictionary with keys that match property names from the COLDP Name class

Methods

NameBundle. add_synonym (*synonym* , *sic = False*)

Register an additional name as a synonym for the accepted name

synonym should contain a dictionary with keys that match property names from the COLDP Name class

Parameters :

- **synonym** – Dictionary mapping COLDP name elements to values for the synonymous name - a name record and a synonym record will be added to the COLDP package for the synonym
- **sic** – Flag to indicate that the name does not follow expected formatting rules for a code-compliant name and that no issues should be logged for this

NameBundle. normalise_name (*name* , *sic = False*)

Ensure that a name record dictionary contains all necessary/appropriate elements

Parameters :

- **name** – Dictionary containing name to be normalised
- **sic** – Flag to indicate that the name does not follow expected formatting rules for a code-compliant name and that no issues should be logged for this

Returns :

Name dictionary updated with extra values

NameBundle. derive_name (*name* , *values* , *sic = False*)

Use supplied values to create new name dictionary with missing elements copied from an existing name dictionary

Parameters :

- **name** – Dictionary containing name on which new name is to be based
- **values** – Dictionary of values to override values in name

Returns :

Name dictionary with supplied values supplemented from name

Index and search

- Index
- Search Page

coldp.IdentifierPolicy

Class

IdentifierPolicy. __init__ (*existing* , *default_prefix* , *required = True* , *volatile = False*)

Internal class to manage ID values in COLDP dataframes.

Parameters :

- **existing** – Pandas series containing current ID values for table - may be empty or None
- **default_prefix** – String prefix to use before numeric ID values if existing ID values are not consistently positive integer values

- **required** – Flag to indicate whether the table must have ID values - if False, the `IdentifierPolicy` will return None unless `existing` already contains ID values
- **volatile** – Flag to indicate if external code may modify ID values while the current COLDP instance is active - if False, the policy and future values will be determined on initialisation, otherwise the policy will be reviewed for each new ID value

Checks any existing values in the series. If none are present, the next ID value will be 1. If all values are integer strings, the next value will be the integer string with a value one higher than the current maximum value. Otherwise ID values will be the concatenation of the prefix (defaulting to the empty string) and the current length of the ID series. Subsequent values will increment by one.

If `required` is False and no current valus exist in the series, the policy will always return None.

If `volatile` is True, the policy will be revised on every invocation of next. Otherwise, the series will only be scanned on initialisation and all policy values will then be fixed.

Methods

IdentifierPolicy. initialise (*existing*)

Internal method to set or refresh policy

Parameters :

existing – Pandas series containing current ID values for table - may be empty or None

Returns :

Tuple comprising the next integer value for the policy and a prefix for use if ID values should not be plain integer strings - one or both values will be None

Provides the variables to determine the next ID value, if any

IdentifierPolicy. next (*existing = None*)

Return next ID value for series

Parameters :

existing – Pandas series containing current ID values for table - may be empty or None - only required if `volatile` is True

Returns :

Next string ID value for policy or None if no ID is required

Returns the next ID value for use in the Series associated with this IdentifierPolicy. Recalculates policy if paramref: *~coldp.IdentifierPolicy.__init__.volatile* is True.

Index and search

- Index
- Search Page

Usage

Usage

The following example illustrates basic usage. It creates a new COLDP instance for the monotypic Lepidoptera family Tridentaformidae, including name and taxon records for the family, genus and species, the original combination for the species, three distribution records for the species, and references for all elements.

Running the same code a second time leaves the data unchanged since it locates and validates the existing records.

In this example, each reference is associated with an ID string provided when the records are created, but all name and taxon ID strings are generated and managed by the COLDP instance.

Adding the synonym to the NameBundle for the species enables the COLDP instance to create the relevant basionymID reference to the current combination.

The example names include a mixture with parsed name elements or a single scientificName element. The final names table includes both formats for all names.

The addition of the distribution records shows the pattern for adding other COLDP classes that reference name (namerelation, typematerial) and taxon (distribution, speciesinteraction) records.

```
# Import COLDP class
from coldp import COLDP

# Default properties for all COLDP Taxon records created by COLDP
instance
taxon_defaults = {
    "kingdom": "Animalia",
    "phylum": "Arthropoda",
    "class": "Insecta",
    "order": "Lepidoptera",
    "status": "established",
}

# Create new COLDP instance with name Tridentaformidae
#
# This would load an existing COLDP instance from a folder named
# Tridentaformidae in the current folder if it already exists
coldp = COLDP("Tridentaformidae", default_taxon_record =
```

```
taxon_defaults)

# Add four COLDP Reference objects
references = coldp.add_references([
    {
        "ID": "Braun_1923",
        "author": "Braun, A.F.",
        "issued": "1923",
        "title": "Microlepidoptera: Notes and New Species",
        "containerTitle":
"Transactions of the American Entomological Society",
        "volume": "49",
        "issue": "2",
        "page": "115-127",
        "link": "https://www.jstor.org/stable/25077087",
    },
    {
        "ID": "Davis_1978",
        "author": "Davis, D.R.",
        "issued": "1978",
        "title":
"Two new genera of North American incurvariine moths (Lepidoptera:
Incurvariidae)",
        "containerTitle": "The Pan-Pacific entomologist",
        "volume": "54",
        "issue": "2",
        "page": "147-153",
        "link": "https://www.biodiversitylibrary.org/page/56100973",
    },
    {
        "ID": "Pohl_et_al_2019",
        "author": "Pohl, G.R., Landry, J.-F., Schmidt, B.C. &
deWaard, J.R.",
        "issued": "2019",
        "title": "Lepidoptera of Canada",
        "containerTitle": "ZooKeys",
        "volume": "819",
        "page": "463-505",
        "link": "https://doi.org/10.3897/zookeys.819.27259",
    },
    {
        "ID": "Regier_et_al_2014",
        "author": "Regier, J.C., Mitter, C., Davis, D.R., Harrison,
T.L., Sohn, J.-C., Cummings, M.P., Zwick, A. & Mitter, K.T.",
        "issued": "2015",
        "title":
"A molecular phylogeny for the oldest (nonditrysian) lineages of
extant Lepidoptera, with implications for classification, comparative
morphology and life-history evolution",
        "containerTitle": "Systematic Entomology",
        "volume": "40",
        "issue": "4",
```

```
        "page": "671–704",
        "link": "https://doi.org/10.1111/syen.12129",
    },
])

# Add COLDP Name and Taxon records for family and get the ID string
for the
# family Taxon record
#
# Name provided as a uninomial
bundle = coldp.start_name_bundle({
    "rank": "family",
    "uninomial": "Tridentaformidae",
    "authorship": "Davis, 2014",
    "referenceID": "Regier_et_al_2014",
    "publishedInPage": "697",
})
coldp.add_names(bundle)
family_id = bundle.accepted_taxon_id

# Add COLDP Name and Taxon records for genus as child of the family
and get
# the ID string for the genus Taxon record
#
# Name provided as a scientificName
bundle = coldp.start_name_bundle({
    "rank": "genus",
    "scientificName": "Tridentaforma",
    "authorship": "Davis, 1978",
    "referenceID": "Davis_1978",
    "publishedInPage": "150",
})
coldp.add_names(bundle, family_id)
genus_id = bundle.accepted_taxon_id

# Add COLDP Name and Taxon records for species as child of the genus
# with original combination as a synonym and get the ID string for
the
# species Taxon record
#
# Accepted name provided as parsed elements. Synonym provided only as
# scientificName.
bundle = coldp.start_name_bundle({
    "rank": "species",
    "genus": "Tridentaforma",
    "specificEpithet": "fuscoleuca",
    "authorship": "(Braun, 1923)",
    "referenceID": "Davis_1978",
    "publishedInPage": "150",
    "publishedInYear": "1978",
})
bundle.add_synonym({
```

```
    "rank": "species",
    "scientificName": "Lampronia fuscoleuca",
    "authorship": "Braun, 1923",
    "referenceID": "Braun_1923",
    "publishedInPage": "127",
})
coldp.add_names(bundle, genus_id)
species_id = bundle.accepted_taxon_id

# Add three distribution records for the species, each with a
reference
for area, referenceID in {"US-CA": "Braun_1923", "CA-AB":
"Pohl_et_al_2019", "CA-BC": "Pohl_et_al_2019"}.items():
    distribution = coldp.add_distribution({
        "taxonID": species_id,
        "area": area,
        "gazetteer": "iso",
        "status": "native",
        "referenceID": referenceID,
    })

# Save the COLDP instance to a Tridentaformidae subfolder in the
current
# folder
coldp.save()

# Load the COLDP instance from the curent folder
coldp = COLDP("Tridentaformidae")

# Display the classification as a text tee
print(coldp.get_text_tree(family_id))

# Show content of dataframes
print(coldp.references)
print(coldp.names)
print(coldp.taxa)
print(coldp.synonyms)
print(coldp.distributions)
```

Index and search

- Index
- Search Page

Created using Sphinx 7.3.7.