Manual Lichen identification

Valerii Goncharuk

Jan 20, 2023

CONTENTS

The application is just being developed and is still very far even from the beta testing stage. However, I hope interested people can help with development, testing, or ideas on how to improve it.

CHAPTER

ONE

USER DOCUMENTATION

1.1 Introduction

1.1.1 About MLI

General idea

Many nature lovers want to know what they observe in nature. Institutes are developing various applications based on artificial intelligence and neural networks to help them. But, at least this applies to fungi and lichens, technology is too far from giving a quality result. In addition, they do not suggest how the naturalist can improve the result of the identification, if it possible.

For the reasons above, those who really want to know have to go back to the pre-computer era, take books and try to guess by the keys.

This application is intended to facilitate the "manual" identification of lichens. It should store all the keys, and unlike paper books, the naturalist no longer has to start with an awkward key. It can be start the identification with any key that is in the database. In addition, the application should help to see those signs that were missed for an accurate identification. For example, to look the reverse side of the lobe in Peltigera, in order to distinguish between forms close to different species.

Taxon Tree

Currently, the taxon tree uses the taxonomy adopted in the GBIF project. The GBIF taxon tree is collected automatically from several sources and is not of high quality. For example, it does not always have scientific names, but only canonical ones, there are taxa that are synonymous with themselves, and etc. However, to date, GBIF is the only platform that has a deployed API and does not severely restrict access to it. So, downloading a tree only takes about a week.

iNaturalist which, in my opinion, has a better taxon tree, albeit often without forms and subspecies, would require from two weeks to a month, as it has a limit of 10,000 API calls per day.

Currently, the database contains 60 965 taxonomic units. Of these, 38.8 thousand are synonyms, 22 thousand are real names. Below is a table that summarizes the taxonomic tree.

rank	Count
kingdom	1
division	2
class	7
order	28
family	126
genus	1199
species	19817
subspecies	258
synonym	28810

1.1.2 Install and run

Note: First of all, you should remember that this application is written in Python 3. And Python versions from 3.6 to 3.10 are supported. So, it is necessary to install its interpreter for work. After installing python you need to install pip.

You can download code from only github now. Download zip archive from github or use git applications to sync with storage. The latter method will allow you to have the latest version of the application and database possible.

If you downloaded zip-file, unzip it to a folder of your choice. Below there are the commands if you decide to use git.

```
git clone https://github.com/tagezi/mli.git
```

In the future, to update the program, you will need to run the command in the application directory *mli*:

```
git pull -r
```

Go to *mli* directory:

```
$ cd mli
```

Install requirements:

```
$ pip install -r requirements.txt
```

Now you are ready to run the application:

```
$ python3 -m mli
```

1.1.3 Licenses

The application is licensed under the GNU General Public License version 3 (GNU GPLv3). All non-software parts including documentation, information stored in databases, photographs, drawings, screenshots are distributed under the Creative Commons Attribution-ShareAlike 4.0 International (CC BY-SA 4.0) license.

In simple terms, you have the right to use, modify and distribute the application and the information in it, if you indicate the source of the information in any reasonable way and do not change the licenses.

1.1.4 Contributors

This list may not be complete, as some authors may not contribute directly to github, but send information and patches to other authors. Also, it may happen that some authors forget to include themselves in the contributor list. So, this is the minimum list of contributors. To get a complete list, you need to do a full revision of authorship.

The list of contributors:

Valerii Goncharuk (aka tagezi)

CHAPTER

TWO

DEV DOCUMENTATION

Developer documentation in development.

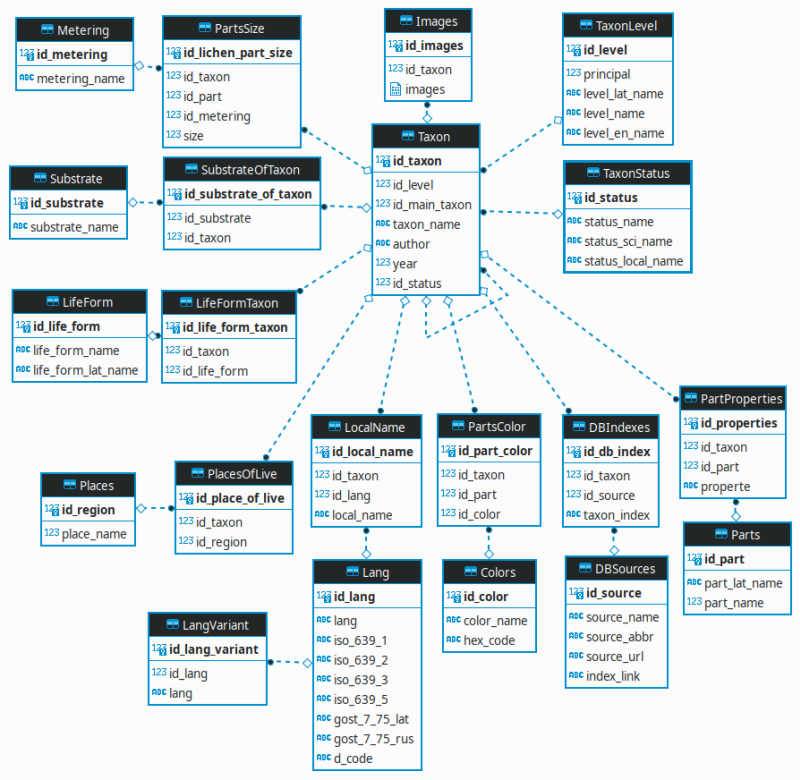
If you decide to help improve this application in any way and have any questions, please contact me. Any contribution is invaluable.

Thank you!

2.1 Database Schema

Below are the relationships in the database.

id_taxon is the taxon name index and the central link for all tables.



2.2 mli

2.2.1 mli package

Subpackages

mli.gui package

Submodules

mli.gui.abstract_classes module

mli.gui.dialog_elements module

class mli.gui.dialog_elements.**ADialogApplyButtons**(*oConnector*, *oParent=None*)

Bases: QDialog

An abstract class that creates a block of Apply, OK, Cancel buttons and reserves action methods for them.

connect_actions_button()

The method of linking signals and button slots.

init_UI_button_block()

Creates a block of buttons for further use in child dialog classes.

onCancel()

The method closes the dialog without saving the data.

onClickApply()

Reserves the Apply dialog button method for future use.

onClickOk()

The method saves the data and closes the dialog In order for the data to be saved, you must override the method onClickApply.

class mli.gui.dialog_elements.**HComboBox**(*sLabel=''*, *oParent=None*)

Bases: QHBoxLayout

Creates a block that units QLabel, QComboBox and QLineEdit. Also, it creates methods that change parameters inside block without direct access.

clear_list()

Clean up a list of QComboBox.

Returns
None

get_text()

The function gets text from QLineEdit of QComboBox.

Returns
Selected text from QComboBox.

Return type
str

get_widget()

set_combo_list(*lItems=None*)

Set up a list of QComboBox.

Parameters
lItems (*list*) – A list of elements for QComboBox.

Returns
None

set_combo_width(*iSize=300*)

Set up width of QComboBox.

Parameters
iSize (*int*) – A number which point to width of QComboBox.

Returns
None

set_label(*sString=''*)

Set up text into Label of block.

Parameters
sString (*str*) – A string which needs to display as Label in the block.

Returns
None

set_text(*sString=''*)

Set up text into QLineEdit of the block.

Parameters
sString (*str*) – A string which display by default in QLineEdit.

Returns
None

class mli.gui.dialog_elements.**HLineEdit**(*sLabel=''*, *iSize=300*, *oParent=None*)

Bases: QHBoxLayout

Creates a block that units QLabel and QLineEdit. Also, it creates methods that change parameters inside block without direct access.

get_text()

The function gets text from QLineEdit.

Returns
Selected text from QLineEdit.

Return type
str

set_label(*sString=''*)

Set up text into Label of block.

Parameters
sString (*str*) – A string which needs to display as Label in the block.

Returns
None

set_line_width(*iSize=300*)

Set up width of QLineEdit.

Parameters
iSize (*int*) – A number which point to width of QComboBox.

Returns
None

set_text(*sString=''*)

Set up text into QLineEdit of the block.

Parameters
sString (*str*) – A string which display by default in QLineEdit.

Returns
None

class mli.gui.dialog_elements.**VComboBox**(*sLabel=''*, *iSize=300*, *oParent=None*)

Bases: QVBoxLayout

Creates a block that units QLabel, QComboBox and QLineEdit. Also, it creates methods that change parameters inside block without direct access.

clear_list()

Clean up a list of QComboBox.

get_text()

The function gets text from QLineEdit of QComboBox.

Returns
Selected text from QComboBox.

Return type
str

get_widget()

set_combo_list(*lItems=None*)

Set up a list of QComboBox.

Parameters
lItems (*list*) – A list of elements for QComboBox.

set_combo_width(*iSize=300*)

Set up width of QComboBox.

Parameters
iSize (*int*) – A number which point to width of QComboBox.

set_label(*sString=''*)

Set up text into Label of block.

Parameters
sString (*str*) – A string which needs to display as Label in the block.

set_text(*sString=''*)

Set up text into QLineEdit of the block.

Parameters
sString (*str*) – A string which display by default in QLineEdit.

class mli.gui.dialog_elements.**VLineEdit**(*sLabel=''*, *iSize=300*, *oParent=None*)

Bases: QVBoxLayout

Creates a block that units QLabel and QLineEdit. Also, it creates methods that change parameters inside block without direct access.

get_text()

The function gets text from QLineEdit.

Returns
Selected text from QLineEdit.

Return type
str

set_label(*sString=''*)

Set up text into Label of block.

Parameters
sString (*str*) – A string which needs to display as Label in the block.

Returns
None

set_line_width(*iSize=300*)

Set up width of QLineEdit.

Parameters
iSize (*int*) – A number which point to width of QComboBox.

Returns
None

set_text(*sString=''*)

Set up text into QLineEdit of the block.

Parameters
sString (*str*) – A string which display by default in QLineEdit.

Returns
None

class mli.gui.dialog_elements.**VTextEdit**(*sLabel=''*, *iSize=300*, *iHeight=120*, *oParent=None*)

Bases: QVBoxLayout

Creates a block that units QLabel and QTextEdit. Also, it creates methods that change parameters inside block without direct access.

clear_text()

The function clears QTextEdit failed.

get_text()

The function gets text from QLineEdit of QTextEdit.

Returns
Selected text from QTextEdit.

Return type
str

set_label(*sString=''*)

Set up text into Label of block.

Parameters
sString (*str*) – A string which needs to display as Label in the block.

Returns
None

set_text(*sString=''*)

Set up text into QTextEdit of the block.

Parameters
sString (*str*) – A string which display by default in QTextEdit.

Returns
None

set_textedit_size(*iWidth=300*, *iHeight=120*)

Set up width of QTextEdit.

Parameters

- **iWidth** (*int*) – A number which point to width of QTextEdit.
- **iHeight** (*int*) – A number which point to height of QTextEdit.

Returns
None

mli.gui.file_dialogs module

class mli.gui.file_dialogs.**OpenDirDialog**(*oParent=None*, *sNameDialog='Dialog'*)

Bases: QFileDialog

exec(*self*) → int

set_dialog()

class mli.gui.file_dialogs.**OpenFileDialog**(*oParent=None*, *dParameter={}*)

Bases: QFileDialog

exec(*self*) → int

mli.gui.help_dialog module

class mli.gui.help_dialog.**About**(*oWidget*)

Bases: QMessageBox

set_dialog()

mli.gui.main_window module

class mli.gui.main_window.**MainWindow**(*sPath*)

Bases: QMainWindow

connect_actions()

It is PyQt5 slots or other words is connecting from GUI element to method or function in program.

create_actions()

Method collect all actions which can do from GUI of program.

get_page_taxon_info(*sTaxonName*)

get_taxon_list()

onDisplayAbout()

Method open dialog window with information about the program.

onEditColor()

onEditColorTaxon()

onEditSubstrate()

onEditSynonym()

onEditTaxon()

onNewColor()

onNewColorTaxon()

onNewSubstrate()

onNewTaxon()

onOpenDB()

onOpenSetting()

onSetStatusBarMessage(*sMassage='Ready'*)

Method create Status Bar on main window of program GUI.

onTaxonInfo()

set_menu_bar()

Method create Menu Bar on main window of program GUI.

mli.gui.message_box module

The module provides message boxes that give hints about incorrect user actions.

***Function*:**

warning_no_synonyms(sName)
warning_lat_name()
warning_restart_app()

warning_this_exist(sThis, sThisName)

***Using*:**

As an example, let's show that the name of the taxon Cladonia, P. Browne already exists.

```
warning_this_exist('taxon name', 'Cladonia, P.Browne')
```

mli.gui.message_box.**warning_lat_name**()

Create a message dialog window with warning that a Latin name of taxon isn't specified.

mli.gui.message_box.**warning_no_synonyms**(*sName*)

mli.gui.message_box.**warning_restart_app**()

Create a message dialog window with warning that app should be restarted.

mli.gui.message_box.**warning_this_exist**(*sThis*, *sThisName*)

Create a dialog window of the message with warning that this exists.

Parameters

- **sThis** (*str*) – A stuff which trying to add.
- **sThisName** (*str*) – The name of stuff which trying to add.

mli.gui.setting_dialog module

class mli.gui.setting_dialog.**SettingDialog**(*oConnector*, *sPathApp*, *oParent=None*)

Bases: *ADialogApplyButtons*

connect_actions()

init_UI()

onClickApply()

Reserves the Apply dialog button method for future use.

onClickOpenFile()

mli.gui.tab_widget module

class mli.gui.tab_widget.**CentralTabWidget**(*oParent*)

Bases: QTabWidget

add_tab(*oWidget=None*, *sName=''*)

connect_actions()

onCloseTab(*index*)

update_tab_name(*iTabIndex=0*, *sTabName='Table 1'*)

mli.gui.taxon_dialogs module

class mli.gui.taxon_dialogs.**ATaxonDialog**(*oConnector*, *oParent=None*)

Bases: *ADialogApplyButtons*

Creates abstract class that contain common elements for Dialogs of taxon.

clean_field()

Clears all fields after use.

connect_actions()

Connects buttons with actions they should perform.

create_level_list(*sTaxon=''*, *bGetAll=None*)

Generates a list of taxon levels depending on a condition. At the first list initialization and using button Apply, all taxon levels are collected, at choosing Main taxon, only those that are below the selected taxon name.

Parameters

- **sTaxon** (*str*) – A name of main taxon.
- **bGetAll** (*bool*) – It is needed to choose all levels.

Returns
list of taxon levels.

Return type
list[str]

create_status_list()

Creates a list of statuses.

Returns
list[str]

create_taxon_list()

Creates a list of taxon names for further use in dialog elements.

Returns
A list in form - (Taxon Rank) Taxon Name

Type
list[str]

fill_combobox()

Fills the fields with the drop-down list during the first initialization and after applying the Apply button.

fill_form(*sSciName*)

init_UI()

initiating a dialog view

onClickApply()

Reserves the Apply dialog button method for future use.

onCurrentMainTaxonChanged(*sTaxon=''*)

The slot that should fire after the taxon name in the Main taxon drop-down list.

Parameters
sTaxon (*str*) – A name of main taxon.

save_(*sSetCol*, *sUpdate*, *sWhere*, *sTable='Taxa'*, *sWhereCol='taxonID'*)

class mli.gui.taxon_dialogs.**EditSynonymDialog**(*oConnector*, *oParent=None*)

Bases: *ATaxonDialog*

connect_actions()

Connects buttons with actions they should perform.

init_UI()

Creating a dialog window.

onCurrentSynonymChanged(*sSynonym*)

onCurrentTaxonNamesChanged(*sTaxonName*)

class mli.gui.taxon_dialogs.**EditTaxonDialog**(*oConnector*, *oParent=None*)

Bases: *ATaxonDialog*

connect_actions()

Connects buttons with actions they should perform.

init_UI()

Creating a dialog window.

onCurrentTaxonNamesChanged(*sTaxonName*)

class mli.gui.taxon_dialogs.**NewTaxonDialog**(*oConnector*, *oParent=None*)

Bases: *ATaxonDialog*

Dialog window which adds information on new taxon.

init_UI()

Creating a dialog window.

onClickApply()

Actions to be taken when adding a new taxon.

Module contents

mli.lib package

Submodules

mli.lib.config module

The module provides an interface for reading and editing the configuration file.

***Classes*:**

ConfigProgram(sFilePath='config.ini')

Using:

There are two ways to use the class.

```
Foo = ConfigProgram()
```

or

```
Foo = ConfigProgram('my_config.ini')
```

Now you can read parameters from configfile and save to it.

```
Parameter = Foo.get_config_value(Section, Option)
```

class mli.lib.config.**ConfigProgram**(*sPathApp*, *sFilePath='config.ini'*)

Bases: ConfigParser

A class for working with a configuration file. Allows you to read from the configuration file and write there.

***Using*:**

```
Foo = ConfigProgram()

# For reading.
Section = 'bar'
Options = 'baz'
Value = Foo.get_config_value(Section, Option)

# For writing.
Value = 'some_string'
Foo.set_config_value (Section, Option, Value)
```

If you only need to create a Section and an Option, you omit the Value.

get_config_value(*sSection*, *sOption*)

The method allows reading from a configuration file.

Parameters

- **sSection** (*str*) – The section in the configuration file to read from.
- **sOption** (*str*) – The option in the configuration file to need reading.

Returns

The value of the specified parameter in the section.

Return type

str

set_config_value(*sSection*, *sOption*, *sValue=''*)

The method allows writing into a configuration file.

Parameters

- **sSection** (*str*) – The section in the configuration file to write to.
- **sOption** (*str*) – The option in the configuration file to write to.
- **sValue** (*str*) – The value to write.

mli.lib.gbif_parser module

The module provides a means to get information from a taxon, as specified in gbif .

function:

gbif_get_children(sGBIF_id) gbif_get_id_from_gbif(sName, sLevel='species') gbif_get_id(oConnector, sName, sLevelEn) gbif_get_many(sURL, sGBIF_id) gbif_get_status_id(oConnector, sStatus) gbif_get_synonyms(sGBIF_id) gbif_get_taxon_info(sGBIF_id, sLevel='species') gbif_get_update(oConnector, iLevel) gbif_is_lichen(dTaxon) gbif_parser_name(sString) gbif_parser_taxon(dData) gbif_parsing_answer(oConnector, lAnswer, sType) gbif_parsing_species(oConnector, dAnswer) gbif_save_species(oConnector, dAnswer, iLevel) gbif_update(oConnector, dAnswer, iTaxonID)

mli.lib.gbif_parser.**gbif_get_children**(*sGBIF_id*)

Generates an api link for obtaining children from the server and
returns a normalized response.

Parameters
sGBIF_id – A key ID of taxon in gbif.

Type
str

Returns
A normalized response.

Return type
list[dict[str, bool, str, str, str, str, str, int]|None]

mli.lib.gbif_parser.**gbif_get_id**(*oConnector*, *sName*, *sLevelEn*)

Checks if the taxon's id exists in the database, and if it doesn't, it gets it from the site.

Parameters

- **oConnector** (SQL) – An instance of the sqlite database api class.
- **sName** (*str*) – A name of the taxon.
- **sLevelEn** (*str*) – A name of the taxon rank in english language.

Returns
ID taxon in gbif.

Return type
int

mli.lib.gbif_parser.**gbif_get_id_from_gbif**(*sName*, *sLevel='species'*)

Looks up a taxon name in the gbif database.

Parameters

- **sName** (*str*) – The name of a taxon.
- **sLevel** (*str*) – The rank of a taxon.

Returns
Key ID of the taxon in gbif.

Return type
str

mli.lib.gbif_parser.**gbif_get_many**(*sURL*, *sGBIF_id*)

Generates an api link for obtaining children from the server and
returns a normalized response.

Parameters

- **sURL** (*str*) – An URL for sending to gbif server.
- **sGBIF_id** – A key ID of taxon in gbif.

Type
str

Returns
A normalized response.

Return type
list[dict[str, bool, str, str, str, str, str, int]|None]

mli.lib.gbif_parser.**gbif_get_status_id**(*oConnector*, *sStatus*)

Returns the status id from the database.

Parameters

- **oConnector** (SQL) – An instance of the sqlite database api class.
- **sStatus** (*str*) – The name of the status, as is customary in gbif.

Returns
the status ID.

Return type
int or bool

mli.lib.gbif_parser.**gbif_get_synonyms**(*sGBIF_id*)

Generates an api link for obtaining synonyms from the server and
returns a normalized response.

Parameters
sGBIF_id – A key ID of taxon in gbif.

Type
str

Returns
A normalized response.

Return type
list[dict[str, bool, str, str, str, str, str, int]|None]

mli.lib.gbif_parser.**gbif_get_taxon_info**(*sGBIF_id*, *sLevel='species'*)

General information about the taxon by its ID.

Parameters

- **sGBIF_id** – A key ID of taxon in gbif.
- **sLevel** (*str*) – A level of the taxon.

Type
str

Returns
A normalized dictionary of taxon information.

Return type
dict[str, bool, str, str, str, str, str, int]|None

mli.lib.gbif_parser.**gbif_get_update**(*oConnector*, *iLevel*)

Allows you to select all names from the database by level, start getting data from gbif and enter information into the database.

Parameters

- **oConnector** (SQL) – An instance of the sqlite database api class.
- **iLevel** (*int*) – ID of a rank in database.

Returns
None

mli.lib.gbif_parser.**gbif_is_lichen**(*dTaxon*)

Checks if the taxon is a lichen.

Parameters
dTaxon (*dict*) – Filed 'result' of answer from gbif.

Returns
True if taxon is lichen, and False if opposite.

Return type
bool

mli.lib.gbif_parser.**gbif_parser_name**(*sString*)

Parsing a name of the taxon separating the canonical name from the name of the author and year, if possible.

Parameters
sString (*str*) – A string with a name of the taxon.

Returns
A canonical name, an author name and a naming year of the taxon.

Return type
list[str, str, int]

mli.lib.gbif_parser.**gbif_parser_taxon**(*dData*)

Selects from the server response the information necessary for further
processing.

Parameters
dData (*dict*) – An answer of server.

Returns
Selection from the server response with the necessary information.

Return type
dict[str, bool, str, str, str, str, str, int]|None

mli.lib.gbif_parser.**gbif_parsing_answer**(*oConnector*, *lAnswer*, *sType*)

Parses the response a list of dictionaries with taxon information.

Parameters

- **oConnector** (SQL) – An instance of the sqlite database api class.

- **lAnswer** (*list[dict[str, bool, str, str, str, str, str, int]|None]*) – A list of dictionaries with taxon information.
- **sType** (*str*) – The first word to output to the string. It makes sense to indicate either 'Synonym' or 'Children'.

Returns
None

mli.lib.gbif_parser.**gbif_parsing_species**(*oConnector*, *dAnswer*)

Specifies whether to make changes to the database.

Parameters

- **oConnector** (SQL) – An instance of the sqlite database api class.
- **dAnswer** (*dict[str, bool, str, str, str, str, str, int]*) – A dictionary with information about the taxon.

Returns
None

mli.lib.gbif_parser.**gbif_save_species**(*oConnector*, *dAnswer*, *iLevel*)

Saving information about the taxon in database.

Parameters

- **oConnector** (SQL) – An instance of the sqlite database api class.
- **dAnswer** (*dict[str, bool, str, str, str, str, str, int]*) – A dictionary with information about the taxon.
- **iLevel** (*int*) – The level's ID in database.

Returns
The taxon's ID in database.

Return type
int or bool

mli.lib.gbif_parser.**gbif_update**(*oConnector*, *dAnswer*, *iTaxonID*)

Updates information in the database about the author and naming year.

Parameters

- **oConnector** (SQL) – An instance of the sqlite database api class.
- **dAnswer** (*dict[str, bool, str, str, str, str, str, int]*) – A dictionary with information about the taxon.
- **iTaxonID** (*int*) – The taxon's ID in database.

Returns
None

mli.lib.log module

mli.lib.log.**start_logging**()

mli.lib.sql module

The module provides an API for working with the database. It creates a multi-level API that can be used in other modules to create requests using a minimum of transmitted data.

Function:
get_columns(sColumns, sConj='AND')

Class:
SQL

Using:
Foo = SQL(_DataBaseFile_)

class mli.lib.sql.**SQL**(*sFileDB*)

Bases: object

Provides interface for working with database from others scripts.

Methods

 # Standard methods.

- __init__ – Method initializes a cursor of sqlite database.
- __del__ – Method closes the cursor of sqlite database.

 # Low level methods.

- export_db – Method exports from db to sql script.
- execute_script – Method imports from slq script to db.
- execute_query – Method execute sql_search query.
- insert_row – Method inserts a record in the database table.
- delete_row – Method deletes a row from the table.
- update – Method updates value(s) in record of the database table.
- select – Method does selection from the table.

 # Average level API.

- sql_get_id: Finds id of the row by value(s) of table column(s).
- sql_get_all: Method gets all records in database table.
- sql_count: Method counts number of records in database table.
- sql_table_clean: Method cleans up the table.

del_garbage(*sSource='GBIF'*)

delete_row(*sTable*, *sColumns=None*, *tValues=None*)

Deletes row in the database table by value(s).

Parameters

- **sTable** (*str or None*) – A table as string in where need to delete row.

- **sColumns** (*str or None*) – Column(s) where the value(s) will be found. (by default, None).
- **tValues** (*tuple or list*) – value(s) as tuple for search of rows. (by default, None).

Returns
True if the deletion is successful, otherwise False.

Return type
bool

execute_query(*sSQL*, *tValues=None*)

Method executes sql script.

Parameters

- **sSQL** (*str*) – SQL query.
- **tValues** (*tuple or list or None*) – value(s) that need to safe inserting into query (by default, None).

Returns
Cursor or bool – True if script execution is successful, otherwise False.

execute_script(*sSQL*)

Method executes sql script.

The main difference from the method is the ability to execute several commands at the same time. For example, using this method, you can restore the database from sql dump.

Parameters
sSQL (*str*) – SQL Script as string.

Returns
True if script execution is successful, otherwise False.

Return type
bool

export_db()

Method exports from db to sql script.

get_all_by_rank(*iRank*)

get_color_id(*sColumn*, *sValue*)

get_full_taxon_list()

get_garbage()

get_id_by_name_author(*tValue*, *sTable='Taxa'*)

get_id_by_name_status(*tValue*)

get_main_taxon(*iTaxonID*)

get_name_author(*aValue*)

get_rank_id(*sColumns*, *sValues*)

get_rank_name(*sColumns*, *iValues*)

get_source_id(*sValue*)

get_status_id(*sValue*, *sColumn='statusLocalName'*)

get_status_taxon(*sSciName*)

get_statuses()

get_substrate_id(*sValue*)

get_synonym_id(*sSciName*)

get_synonyms(*iValue*)

get_taxon_children(*iID*, *sStatus*)

get_taxon_db_link(*iID*)

get_taxon_id(*sSciName*, *sAuthor=''*)

get_taxon_info(*sName*)

get_taxon_list(*sStatus*)

get_taxon_rank(*sSciName*)

insert_row(*sTable*, *sColumns*, *tValues*)

Inserts a record in the database table.

Parameters

- **sTable** (*str*) – Table name as string.
- **sColumns** (*str*) – Columns names of the table by where needs inserting.
- **tValues** (*tuple or list*) – Value(s) as tuple for inserting.

Returns

ID of an inserted row if the insert was successful. Otherwise, False.

Return type

str or bool

insert_taxon(*sName*, *sAuthor*, *iYear*, *PublishedIn*, *iRank*, *iMainTax*, *iStatus*)

select(*sTable*, *sGet*, *sWhere=''*, *tValues=''*, *sConj=''*, *sFunc=''*)

Looks for row by value(s) in table column(s).

Parameters

- **sTable** (*str*) – Table name as string.
- **sGet** (*str*) – Name of the column of the table, which will be returned.
- **sWhere** (*str or None*) – Names of columns of the table, by which to search (by default, empty).
- **sConj** (*str or None*) – The one from 'AND' or 'OR' operator condition. By default, is used 'AND'.
- **tValues** (*tuple or list or None*) – Value(s) as tuple for search (by default, empty).
- **sFunc** (*str or None*) – Function name of sqlite, which need to apply (by default, empty). Note: Now, you can use only two sqlite functions: Count and DISTINCT.

Returns
Cursor or bool – Cursor object within rows that was found, or False, if the row not found.

`sql_count`(*sTable*)

Counts number of records in database table.

Parameters
sTable (*str*) – Table name as string where records should be count.

Returns
Number of found records.

Return type
int or bool

`sql_get_all`(*sTable*)

Gets all records in database table.

Parameters
sTable (*str*) – Table name as string where records should be received.

Returns
Tuple of all rows of table.

Return type
tuple or bool

`sql_get_id`(*sTable*, *sID*, *sWhere*, *tValues*, *sConj=''*)

`sql_get_values`(*sTable*, *sID*, *sWhere*, *tValues*, *sConj=''*)

Looks for ID of the row by value(s) of table column(s).

Parameters

- **sTable** (*str*) – Table name as string.
- **sID** (*str*) – Name of the column of the table by which to search.
- **sWhere** (*str*) – Names of columns of the table by which to search.
- **tValues** (*tuple or list*) – Value(s) as tuple for search.
- **sConj** (*str or None*) – The one from 'AND' or 'OR' operator condition. By default, is used 'AND'.

Returns
ID as Number in the row cell, or 0, if the row not found.

Return type
list or bool

`sql_table_clean`(*lTable*)

Cleans up the table.

Parameters
lTable (*tuple or list*) – Table names as list or tuple of string, or table name as string where cleaning is need to do.

Returns
True, if execution is successful. Otherwise, False. Note: False is returned even if cleaning the last table in the tuple was not successful.

Return type
bool

update(*sTable*, *sSetUpdate*, *sWhereUpdate*, *tValues*)

Updates value(s) in the record of the database table.

Parameters

- **sTable** (*str*) – A Table as string where update is need to do.
- **sSetUpdate** (*str*) – Column(s) where the value are writen.
- **sWhereUpdate** (*str*) – A column where values correspond to the required.
- **tValues** (*tuple or list*) – value(s) as tuple for search corresponding rows.

Returns

True if the insert was successful, otherwise False.

Return type

bool

mli.lib.sql.**check_connect_db**(*oConnector*, *sBasePath*, *sDBDir*)

Checks for the existence of a database and if it does not find it, then
creates it with default values.

Parameters

- **oConnector** (SQL) – Instance attribute of SQL.
- **sBasePath** (*str*) – A path of the executed script.
- **sDBDir** (*str*) – A dir when database is by default.

Returns

None

mli.lib.sql.**get_columns**(*sColumns*, *sConj='AND'*)

The function of parsing a string, accepts a list of table columns separated by commas and returns this list with '=? AND' or '=? OR' as separator.

Parameters

- **sColumns** (*str*) – A string with a list of table columns separated by commas.
- **sConj** (*str or None*) – The one from 'AND' or 'OR' operator condition. By default, is used 'AND'.

Returns

The string with a list of table columns separated by '=? AND' or '=? OR'.

Return type

str

mli.lib.sql.**get_increase_value**(*sColumns*, *tValues*)

Checks counting elements of values, and if them fewer,
then makes them equal.

Note:

In the rison that tuple can't be multiplied on flot, the process of increasing the tuple becomes somewhat resource-intensive. So, tValues should be consisting of one element.

Parameters

- **sColumns** (*str*) – Colum(s) in query.

- **tValues** (*tuple or list*) – Values should be specified in the request.

Returns
A tuple with values, which equal to sColumns.

Return type
list

mli.lib.str module

The module contains a collection of functions for solving routine tasks with strings.

class mli.lib.str.**HTMLDoc**

Bases: object

get_doc()

set_is_synonym(*sName*, *sAuthor*, *sMainName*, *sMainAuthor*)

set_link(*sSource*, *sLink*, *sIndex*)

set_no_data(*sNoData*)

set_rang_name(*sRank*, *sName*, *sAuthor*)

set_string(*String*)

set_title_chart(*sTitle*)

set_title_doc(*sRank*, *sName*, *sAuthor*)

mli.lib.str.**str_get_file_patch**(*sDir*, *sFile*)

Concatenates file path and file name based on OS rules.

Parameters

- **sDir** – String with a patch to a file.
- **sFile** – String with a filename.

Returns
Patch to file based on OS rules.

mli.lib.str.**str_get_html_name**(*sCanonicalName*, *sAuthor=''*)

Inserts html tags into the taxon scientific name so that the canonical
name is written in italics.

Parameters

- **sCanonicalName** (*str*) – A canonical name of the taxon.
- **sAuthor** (*str*) – Author(s) of the taxon.

Returns
Formatted string.

Return type
str

mli.lib.str.**str_get_path**(*sFullFile*)

Splits a path to path and file name.

Parameters
sFullFile (*str*) – Path with filename.

Returns
The path to file.

Return type
str

mli.lib.str.**str_sep_comma**(*sString*)

Separates a string by comma to list.

Parameters
sString (*str*) – A string that needs to separate.

Returns
A separated string by comma.

Return type
list or None

mli.lib.str.**str_sep_dot**(*sString*)

Separates a string by dot to list.

Parameters
sString (*str*) – A string that needs to separate.

Returns
A separated string by dot.

Return type
list or None

mli.lib.str.**str_sep_name_taxon**(*sString*)

Splits the string into taxon name and author, taking a string of the
form '(rank) Taxon_name, authors'. It is permissible to indicate authors separated by commas, in brackets, using the '&' symbol.

Parameters
sString (*str*) – A string that needed to separate.

Returns
A canonical form of taxon name and a string with the authors. Returns empty instead of author if no author was specified in the string.

Return type
list[str, str|None]

mli.lib.str.**str_text_to_list**(*sString*)

Module contents

Submodules

mli.version module

Module contents

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

m

INDEX

H

I

M

N

O

S

U

V

W