astrodbkit Documentation

Release 0.6.6

Joe Filippazzo

Jan 19, 2018

Contents

This documentation describes a toolkit of classes, methods, and functions useful for CRUD operations and analysis of data from an SQL database.

CHAPTER 1

Getting Started

To install **astrodbkit**, do:

```
pip install astrodbkit
```

Alternatively, you can update your existing installation with:

```
pip install --upgrade astrodbkit
```

CHAPTER 2

Creating a Database

To create a database from scratch, do:

```
from astrodbkit import astrodb
dbpath = '/desired/path/to/my_new_database.db'
astrodb.create_database(dbpath)
```

Alternatively, you can download and use the BDNYC Database, which contains the astrometry, photometry, and spectra for the 198 objects in the Filippazzo et al. (2015) sample.

Note: For access to the full dataset, an email request must be made to a BDNYC group admin.

CHAPTER 3

Accessing the Database

To start using the database, launch Python, import the module, then initialize the database with the `astrodb.Database()` class like so:

```
from astrodbkit import astrodb
db = astrodb.Database(dbpath)
```

You are now ready to use the database.

Note: The path to the database can either be the binary database file (typically ending in .db) or an .sql file that contains the schema for the database. In the later case, the database is constructed by referring to the schema and the directory where the tables of data are located (by default, 'tabledata') and a .db file is created. You can use the `save()` method to output a database to a schema file and the individual tables to a specified directory. This workflow can better facilitate version control.

CHAPTER 4

Querying the Database

4.1 Getting Help

It can be daunting to start using the database without a lot of prior knowledge of the contents of the database or the functionality of astrodbkit. For this purpose we have created two utility methods:

```
db.info()
```

Will list the names of the files that have been loaded as well as the contents of the database: every table and the number of sources in it.

And:

```
db.help()
```

Will give a brief overview of what `astrodb.Database()` is and summarizes the more widely used methods. More details on each method can be obtained by using Python's help system, for example:

```
help(db.query)
```

4.2 Specialized Searches

Now that you have the database loaded, you'll want to get some information out of it. There are a variety of ways to extract information with astrodbkit.

The schema for any table can be quickly examined with the `schema()` method:

```
db.schema('sources')
```

You can see an inventory of all data for a specific source by passing an integer id to the `inventory()` method:

```
data = db.inventory(86)
```

This will retrieve the data across all tables with the specified source_id for visual inspection. Setting *fetch=True* will return the data as a dictionary of Astropy tables so that table and column keys can be used to access the results. For example:

```
data['photometry'][['band','magnitude','magnitude_unc']]
```

will return a table of the band, magnitude and uncertainty for all records in the sources table with that source_id.

You can search any table in the database with the `identify()` method by supplying a string, integer, or (ra,dec) coordinates along with the table to search. For example, if I want to find all the records in the SOURCES table in the HR 8799 system:

```
db.search('8799', 'sources')
```

Or all the papers published by Joe Filippazzo:

```
db.search('Fili', 'publications')
```

When supplying coordinates, you can also specify the search *radius* to use, in degrees:

```
db.search((338.673, 40.694), 'sources', radius=5)
```

The `references()` method can be used to search for all entries that match the publication record. For example:

```
db.references('Cruz03')
```

4.3 General Queries

You can also pass SQL queries wrapped in double-quotes (") to the `query()` method:

```
data = db.query( "SQL_query_goes_here" )
```

For example, you can get an Astropy table of all the records with a spectral type of L2 with:

```
db.query("SELECT * FROM spectral_types WHERE spectral_type=12", fmt='table')
```

By default, this returns the data as a list of arrays. Alternative options for the fmt flag include *'dict'* for a list of Python dictionaries, *'table'* for an Astropy Table, and *'pandas'* for a pandas DataFrame.

Here is a detailed post about how to write a SQL query.

For more general SQL commands beyond SELECT and PRAGMA, you can use the `modify()` method:

```
db.modify("UPDATE spectra SET wavelength_units='A' WHERE id=4")
```

CHAPTER 5

Adding Data

There are two main ways to add data to a database with **astrodbkit**: by passing a properly formatted ascii file or by passing the data directly in a list of lists.

To add data from a file, you want to create a file with the following format:

```
ra|dec|publication_id
123|-34|5
```

Each entry should be its own row, with the first row denoting the columns to be populated. Note that the column names in the ascii file need not be in the same order as the table. Also, only the column names that match will be added and non-matching or missing column names will be ignored. Assuming this file is called **data.txt** in the working directory, we can add this new data to the SOURCES table with:

```
db.add_data('data.txt', 'sources', delim='|')
```

To add the same data without creating the file, you would do the following:

```
data = [['ra', 'dec', 'publication_id'],[123, -34, 5]]
db.add_data(data, 'sources')
```

CHAPTER 6

Saving Results

The `query()` method provides an option to export your query to a file:

```
db.query(my_query, export='results.txt')
```

By default, this will save the results to an ascii file.

VOTables are another way to store data in a format that can be read by other programs, such as TOPCAT. The `votools` module can generate a VOTable from your SQL query. This has been integrated to be called directly by the `query()` method when the filename ends in **.xml** or **.vot**. For example:

```
from astrodbkit import astrodb
db = astrodb.Database('/path/to/database')
txt = 'SELECT s.id, s.ra, s.dec, s.shortname, p.source_id, p.band, p.magnitude FROM␣
↪sources as s ' \
      'JOIN photometry as p ON s.id=p.source_id WHERE s.dec<=-10 AND (p.band IN ("J",
↪"H","Ks","W1"))'
data = db.query(txt, export='votable.xml')
```

You can import and call votools directly, which has additional options you can set.

Note: Special characters (such as accents or greek letters) can cause astropy and thus file output to fail in Python 2. Python 3 handles this differently and will not fail in this instance.

CHAPTER 7

Saving the Full Database

If changes have been made to the database, such as by adding new data or modifying existing entries, you will want to use the `save()` method to dump the contents of the database to ascii files. `save()` writes a schema file and outputs all tables to individual files in a directory of your choice (by default, 'tabledata'). This directory and the schema file can be version controlled with, for example, git, to facilitate tracking changes in the database.

When finished working with the database, the `close()` method will close the connection.

Note: `close()` will prompt to save the database to the default directory.

CHAPTER 8

Contents

8.1 astrodb module

class astrodbkit.astrodb.**Database**(*dbpath*, *directory=u''*)

add_changelog(*user=u''*, *mod_tables=u''*, *user_desc=u''*)
Add an entry to the changelog table. This should be run when changes or edits are done to the database.

Parameters

- **user** (*str*) – Name of the person who made the edits
- **mod_tables** (*str*) – Table or tables that were edited
- **user_desc** (*str*) – A short message describing the changes

add_data(*data*, *table*, *delimiter=u'|'*, *bands=u''*, *clean_up=True*, *verbose=False*)
Adds data to the specified database table. Column names must match table fields to insert, however order and completeness don't matter.

Parameters

- **data** (*str, array-like, astropy.table.Table*) – The path to an ascii file, array-like object, or table. The first row or element must be the list of column names
- **table** (*str*) – The name of the table into which the data should be inserted
- **delimiter** (*str*) – The string to use as the delimiter when parsing the ascii file
- **bands** (*sequence*) – Sequence of band to look for in the data header when digesting columns of multiple photometric measurements (e.g. ['MKO_J','MKO_H','MKO_K']) into individual rows of data for database insertion
- **clean_up** (*bool*) – Run self.clean_up()
- **verbose** (*bool*) – Print diagnostic messages

add_foreign_key (*table*, *parent*, *key_child*, *key_parent*, *verbose=True*)
Add foreign key (**key_parent** from **parent**) to **table** column **key_child**

Parameters

- **table** (*string*) – The name of the table to modify. This is the child table.
- **parent** (*string or list of strings*) – The name of the reference table. This is the parent table.
- **key_child** (*string or list of strings*) – Column in **table** to set as foreign key. This is the child key.
- **key_parent** (*string or list of strings*) – Column in **parent** that the foreign key refers to. This is the parent key.
- **verbose** (*bool, optional*) – Verbose output

clean_up (*table*, *verbose=False*)
Removes exact duplicates, blank records or data without a *source_id* from the specified **table**. Then finds possible duplicates and prompts for conflict resolution.

Parameters

- **table** (*str*) – The name of the table to remove duplicates, blanks, and data without source attributions.
- **verbose** (*bool*) – Print out some diagnostic messages

close (*silent=False*)
Close the database and ask to save and delete the file

Parameters **silent** (*bool*) – Close quietly without saving or deleting (Default: False).

get_bibtex (*id*, *fetch=False*, *table=u'publications'*)
Grab bibtex entry from NASA ADS

Parameters

- **id** (*int or str*) – The id or shortname from the PUBLICATIONS table to search
- **fetch** (*bool*) – Whether or not to return the bibtex string in addition to printing (default: False)
- **table** (*str*) – Table name, defaults to publications

Returns **bibtex** – If fetch=True, return the bibtex string

Return type str

help ()
See a quick summary of the most useful methods in astrodb.Database

info ()
Prints out information for the loaded database, namely the available tables and the number of entries for each.

inventory (*source_id*, *fetch=False*, *fmt=u'table'*)
Prints a summary of all objects in the database. Input string or list of strings in **ID** or **unum** for specific objects.

Parameters

- **source_id** (*int*) – The id from the SOURCES table whose data across all tables is to be printed.

- **fetch** (*bool*) – Return the results.
- **fmt** (*str*) – Returns the data as a dictionary, array, or astropy.table given 'dict', 'array', or 'table'

Returns **data_tables** – Returns a dictionary of astropy tables with the table name as the keys.

Return type dict

lookup (*criteria*, *table*, *columns=u"*)
Returns a table of records from *table* the same length as *criteria* with the best match for each element.

Parameters

- **criteria** (*sequence*) – The search criteria
- **table** (*str*) – The table to search
- **columns** (*sequence*) – The column name in the sources table to search

Returns **results** – A sequence the same length as objlist with source_ids that correspond to successful matches and blanks where no matches could be made

Return type sequence

merge (*conflicted*, *tables=[]*, *diff_only=True*)
Merges specific **tables** or all tables of **conflicted** database into the master database.

Parameters

- **conflicted** (*str*) – The path of the SQL database to be merged into the master.
- **tables** (*list (optional)*) – The list of tables to merge. If None, all tables are merged.
- **diff_only** (*bool*) – If True, only prints the differences of each table and doesn't actually merge anything.

modify (*SQL*, *params=u"*, *verbose=True*)
Wrapper for CRUD operations to make them distinct from queries and automatically pass commit() method to cursor.

Parameters

- **SQL** (*str*) – The SQL query to execute
- **params** (*sequence*) – Mimics the native parameter substitution of sqlite3
- **verbose** (*bool*) – Prints the number of modified records

output_spectrum (*spectrum*, *filepath*, *header={}*)
Prints a file of the given spectrum to an ascii file with specified filepath.

Parameters

- **spectrum** (*int, sequence*) – The id from the SPECTRA table or a [w,f,e] sequence
- **filepath** (*str*) – The path of the file to print the data to.
- **header** (*dict*) – A dictionary of metadata to add of update in the header

plot_spectrum (*spectrum_id*, *table=u'spectra'*, *column=u'spectrum'*, *overplot=False*, *color=u'b'*, *norm=False*)
Plots a spectrum from the given column and table

Parameters

- **spectrum_id** (*int*) – The id from the table of the spectrum to plot.

- **`overplot`** (*`bool`*) – Overplot the spectrum
- **`table`** (*`str`*) – The table from which the plot is being made
- **`column`** (*`str`*) – The column with SPECTRUM data type to plot
- **`color`** (*`str`*) – The color used for the data
- **`norm`** (*`bool, sequence`*) – True or (min,max) wavelength range in which to normalize the spectrum

`query`(*SQL*, *params=u''*, *fmt=u'array'*, *fetch=u'all'*, *unpack=False*, *export=u''*, *verbose=False*, *use_converters=True*)

Returns data satisfying the provided **SQL** script. Only SELECT or PRAGMA statements are allowed. Results can be returned in a variety of formats. For example, to extract the ra and dec of all entries in SOURCES in astropy.Table format one can write:

data = db.query('SELECT ra, dec FROM sources', fmt='table')

For more general SQL statements, see the modify() method.

Parameters

- **`SQL`** (*`str`*) – The SQL query to execute
- **`params`** (*`sequence`*) – Mimics the native parameter substitution of sqlite3
- **`fmt`** (*`str`*) – Returns the data as a dictionary, array, astropy.table, or pandas.Dataframe given 'dict', 'array', 'table', or 'pandas'
- **`fetch`** (*`str`*) – String indicating whether to return **all** results or just **one**
- **`unpack`** (*`bool`*) – Returns the transpose of the data
- **`export`** (*`str`*) – The file path of the ascii file to which the data should be exported
- **`verbose`** (*`bool`*) – print the data as well
- **`use_converters`** (*`bool`*) – Apply converters to columns with custom data types

Returns **result** – The result of the database query

Return type (array,dict,*table*)

`references`(*criteria*, *publications=u'publications'*, *column_name=u'publication_shortname'*, *fetch=False*)

Do a reverse lookup on the **publications** table. Will return every entry that matches that reference.

Parameters

- **`criteria`** (*`int or str`*) – The id from the PUBLICATIONS table whose data across all tables is to be printed.
- **`publications`** (*`str`*) – Name of the publications table
- **`column_name`** (*`str`*) – Name of the reference column in other tables
- **`fetch`** (*`bool`*) – Return the results.

Returns **data_tables** – Returns a dictionary of astropy tables with the table name as the keys.

Return type dict

`save`(*directory=None*)

Dump the entire contents of the database into the tabledata directory as ascii files

`schema`(*table*)

Print the table schema

Parameters **`table`** (*`str`*) – The table name

`search`(*criterion*, *table*, *columns=u"*, *fetch=False*, *radius=0.016666666666666666*, *use_converters=False*, *sql_search=False*)

General search method for tables. For (ra,dec) input in decimal degrees, i.e. (12.3456,-65.4321), returns all sources within 1 arcminute, or the specified radius. For string input, i.e. 'vb10', returns all sources with case-insensitive partial text matches in columns with 'TEXT' data type. For integer input, i.e. 123, returns all exact matches of columns with INTEGER data type.

Parameters

- **`criterion`** (*`(str, int, sequence, tuple)`*) – The text, integer, coordinate tuple, or sequence thereof to search the table with.
- **`table`** (*`str`*) – The name of the table to search
- **`columns`** (*`sequence`*) – Specific column names to search, otherwise searches all columns
- **`fetch`** (*`bool`*) – Return the results of the query as an Astropy table
- **`radius`** (*`float`*) – Radius in degrees in which to search for objects if using (ra,dec). Default: 1/60 degree
- **`use_converters`** (*`bool`*) – Apply converters to columns with custom data types
- **`sql_search`** (*`bool`*) – Perform the search by coordinates in a box defined within the SQL commands, rather than with true angular separations. Faster, but not a true radial search.

`show_image`(*image_id*, *table=u'images'*, *column=u'image'*, *overplot=False*, *cmap=u'hot'*, *log=False*)

Plots a spectrum from the given column and table

Parameters

- **`image_id`** (*`int`*) – The id from the table of the images to plot.
- **`overplot`** (*`bool`*) – Overplot the image
- **`table`** (*`str`*) – The table from which the plot is being made
- **`column`** (*`str`*) – The column with IMAGE data type to plot
- **`cmap`** (*`str`*) – The colormap used for the data

`snapshot`(*name_db=u'export.db'*, *version=1.0*)

Function to generate a snapshot of the database by version number.

Parameters

- **`name_db`** (*`string`*) – Name of the new database (Default: export.db)
- **`version`** (*`float`*) – Version number to export (Default: 1.0)

`table`(*table*, *columns*, *types*, *constraints=u"*, *pk=u"*, *new_table=False*)

Rearrange, add or delete columns from database **table** with desired ordered list of **columns** and corresponding data **types**.

Parameters

- **`table`** (*`sequence`*) – The name of the table to modify
- **`columns`** (*`list`*) – A sequence of the columns in the order in which they are to appear in the SQL table

- **types** (*sequence*) – A sequence of the types corresponding to each column in the columns list above.
- **constraints** (*sequence (optional)*) – A sequence of the constraints for each column, e.g. '', 'UNIQUE', 'NOT NULL', etc.
- **pk** (*string or list*) – Name(s) of the primary key(s) if other than ID
- **new_table** (*bool*) – Create a new table

class astrodbkit.astrodb.**Image**(*data*, *header=u''*, *path=u''*)

exception astrodbkit.astrodb.**InputError**(*message*)

Bases: exceptions.Exception

Exception raised for errors in the input.

message -- explanation of the error

class astrodbkit.astrodb.**Spectrum**(*data*, *header=u''*, *path=u''*)

astrodbkit.astrodb.**adapt_array**(*arr*)

Adapts a Numpy array into an ARRAY string to put into the database.

Parameters arr (*array*) – The Numpy array to be adapted into an ARRAY type that can be inserted into a SQL file.

Returns The adapted array object

Return type ARRAY

astrodbkit.astrodb.**ang_sep**(*row*, *ra1*, *dec1*)

Calculate angular separation between two coordinates Uses Vicenty Formula (http://en.wikipedia.org/wiki/Great-circle_distance) and adapts from astropy's SkyCoord Written to be used within the Database.search() method

Parameters

- **row** (*dict, pandas Row*) – Coordinate structure containing ra and dec keys in decimal degrees
- **ra1** (*float*) – RA to compare with, in decimal degrees
- **dec1** (*float*) – Dec to compare with, in decimal degrees

Returns

Return type Angular distance, in degrees, between the coordinates

astrodbkit.astrodb.**clean_header**(*header*)

astrodbkit.astrodb.**convert_array**(*array*)

Converts an ARRAY string stored in the database back into a Numpy array.

Parameters array (*ARRAY*) – The array object to be converted back into a Numpy array.

Returns The converted Numpy array.

Return type array

astrodbkit.astrodb.**convert_image**(*File*, *verbose=False*)

Converts a IMAGE data type stored in the database into a data cube

Parameters

- **File** (*str*) – The URL or filepath of the file to be converted into arrays.
- **verbose** (*bool*) – Whether or not to display some diagnostic information (Default: False)

Returns The converted image

Return type sequence

`astrodbkit.astrodb.`**`convert_spectrum`**(*File*, *verbose=False*)

Converts a SPECTRUM data type stored in the database into a (W,F,E) sequence of arrays.

Parameters

- **`File`** (*`str`*) – The URL or filepath of the file to be converted into arrays.
- **`verbose`** (*`bool`*) – Whether or not to display some diagnostic information (Default: False)

Returns The converted spectrum.

Return type sequence

`astrodbkit.astrodb.`**`create_database`**(*dbpath*, *schema=u"*, *overwrite=True*)

Create a new database at the given dbpath

Parameters

- **`dbpath`** (*`str`*) – The full path for the new database, including the filename and .db file extension.
- **`schema`** (*`str`*) – The path to the .sql schema for the database
- **`overwrite`** (*`bool`*) – Overwrite dbpath if it already exists

`astrodbkit.astrodb.`**`pprint`**(*data*, *names=u"*, *title=u"*, *formats={}*)

Prints tables with a bit of formatting

Parameters

- **`data`** (*`(sequence, dict, table)`*) – The data to print in the table
- **`names`** (*`sequence`*) – The column names
- **`title`** (*`str (optional)`*) – The title of the table
- **`formats`** (*`dict`*) – A dictionary of column:format values

`astrodbkit.astrodb.`**`scrub`**(*data*, *units=False*)

For input data [w,f,e] or [w,f] returns the list with NaN, negative, and zero flux (and corresponding wavelengths and errors) removed.

8.2 astrocat module

8.3 votools module

`astrodbkit.votools.`**`dict_tovot`**(*tabdata*, *tabname='votable.xml'*, *phot=False*, *binary=True*)

Converts dictionary table **tabdata** to a VOTable with name **tabname**

Parameters

- **`tabdata`** (*`list`*) – SQL query dictionary list from running query_dict.execute()
- **`tabname`** (*`str`*) – The name of the VOTable to be created
- **`phot`** (*`bool`*) – Parameter specifying if the table contains photometry to be merged
- **`binary`** (*`bool`*) – Parameter specifying if the VOTable should be saved as a binary. This is necessary for tables with lots of text columns.

`astrodbkit.votools.`**`photaddline`**(*tab*, *sourceid*)

Loop through the dictionary list **tab** creating a line for the source specified in **sourceid**

Parameters

- **`tab`** – Dictionary list of all the photometry data
- **`sourceid`** – ID of source in the photometry table (source_id)

Returns **tmpdict** – Dictionary with all the data for the specified source

Return type dict

`astrodbkit.votools.`**`photparse`**(*tab*)

Parse through a photometry table to group by source_id

Parameters **`tab`** (*`list`*) – SQL query dictionary list from running query_dict.execute()

Returns **newtab** – Dictionary list after parsing to group together sources

Return type list

`astrodbkit.votools.`**`table_add`**(*tab*, *data*, *col*)

Function to parse dictionary list **data** and add the data to table **tab** for column **col**

Parameters

- **`tab`** (*`Table class`*) – Table to store values
- **`data`** (*`list`*) – Dictionary list from the SQL query
- **`col`** (*`str`*) – Column name (ie, dictionary key) for the column to add

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index