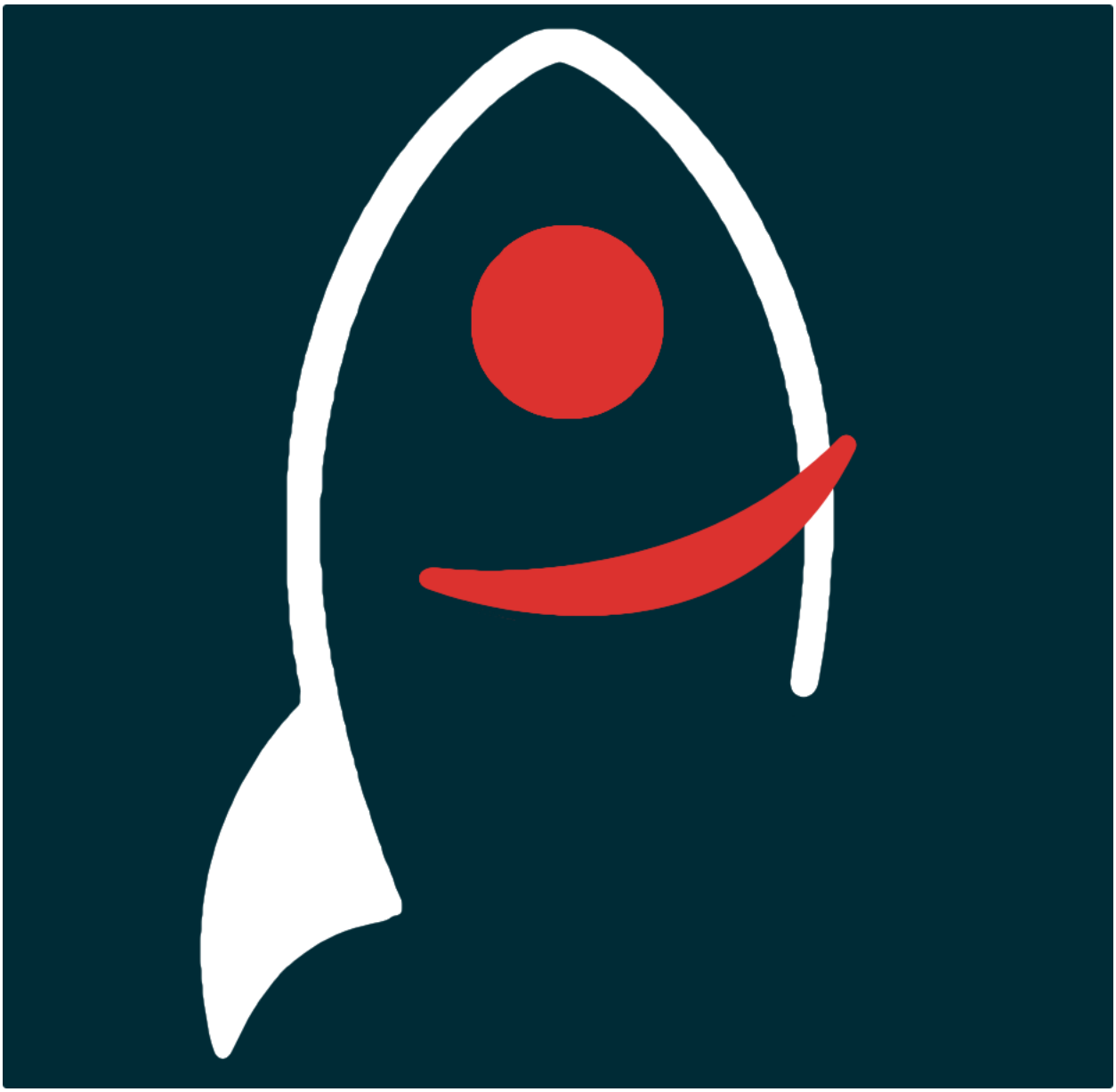


sherlock Documentation

Release v2.0.2

Dave Young

2020

Getting Started

1 Installation **3**
- 1.1 Development . . . 3
  - 1.1.1 Sublime Snippets . . . 4
- 1.2 Issues . . . 4

2 Command-Line Usage **5**

3 Documentation **7**

4 Sherlock Tutorial **9**
- 4.1 Initialisation and Setup . . . 9
  - 4.1.1 Populating Sherlock's Settings File . . . 9
    - 4.1.1.1 Database Settings . . . 9
    - 4.1.1.2 The Search Algorithm . . . 10
- 4.2 Classifying Transients . . . 11
  - 4.2.1 A Single Transient Classification . . . 11
  - 4.2.2 Classifying Transients in a Transient Database . . . 11
  - 4.2.3 The Classification Workflow . . . 11
    - 4.2.3.1 Synonyms vs Associations . . . 11
    - 4.2.3.2 NED Stream Updater . . . 12
    - 4.2.3.3 Search Algoritm . . . 13
    - 4.2.3.4 Angular Separation Search . . . 13
    - 4.2.3.5 Physical Separation Search . . . 14
    - 4.2.3.6 Classification Rankings . . . 14
- 4.3 Sherlock's Catalogue Database . . . 15
  - 4.3.1 Database Table Naming Scheme . . . 15
    - 4.3.1.1 Table Classes . . . 15
    - 4.3.1.2 Versioning . . . 15
  - 4.3.2 Maintainance and Updates of Catalogues Database . . . 15
  - 4.3.3 Updating Catalogues and Adding New Catalogues to the Database . . . 16

5 Installation **19**
- 5.1 Development . . . 19
  - 5.1.1 Sublime Snippets . . . 20
- 5.2 Issues . . . 20

6 Command-Line Usage **21**

7 Documentation **23**

8 Sherlock Tutorial **25**
8.1 Initialisation and Setup . . . 25
8.1.1 Populating Sherlock's Settings File . . . 25
8.1.1.1 Database Settings . . . 25
8.1.1.2 The Search Algorithm . . . 26
8.2 Classifying Transients . . . 27
8.2.1 A Single Transient Classification . . . 27
8.2.2 Classifying Transients in a Transient Database . . . 27
8.2.3 The Classification Workflow . . . 27
8.2.3.1 Synonyms vs Associations . . . 27
8.2.3.2 NED Stream Updater . . . 28
8.2.3.3 Search Algoritm . . . 29
8.2.3.4 Angular Separation Search . . . 29
8.2.3.5 Physical Separation Search . . . 30
8.2.3.6 Classification Rankings . . . 30
8.3 Sherlock's Catalogue Database . . . 31
8.3.1 Database Table Naming Scheme . . . 31
8.3.1.1 Table Classes . . . 31
8.3.1.2 Versioning . . . 31
8.3.2 Maintainance and Updates of Catalogues Database . . . 31
8.3.3 Updating Catalogues and Adding New Catalogues to the Database . . . 32
8.3.3.1 Subpackages . . . 33
8.3.3.1.1 sherlock.commonutils (*subpackage*) . . . 33
8.3.3.1.2 sherlock.imports (*subpackage*) . . . 33
8.3.3.2 Classes . . . 34
8.3.3.2.1 sherlock.catalogue_conesearch (*class*) . . . 34
8.3.3.2.2 sherlock.commonutils.update_wiki_pages (*class*) . . . 37
8.3.3.2.3 sherlock.database (*class*) . . . 39
8.3.3.2.4 sherlock.database_cleaner (*class*) . . . 41
8.3.3.2.5 sherlock.imports._base_importer (*class*) . . . 44
8.3.3.2.6 sherlock.imports.ifs (*class*) . . . 45
8.3.3.2.7 sherlock.imports.marshall (*class*) . . . 47
8.3.3.2.8 sherlock.imports.ned (*class*) . . . 49
8.3.3.2.9 sherlock.imports.ned_d (*class*) . . . 53
8.3.3.2.10 sherlock.imports.veron (*class*) . . . 57
8.3.3.2.11 sherlock.transient_catalogue_crossmatch (*class*) . . . 60
8.3.3.2.12 sherlock.transient_classifier (*class*) . . . 66
8.3.3.3 Functions . . . 75
8.3.3.3.1 sherlock.commonutils.get_crossmatch_catalogues_column_map (*function*) . . . 76
8.4 Indexes . . . 78
8.5 Todo . . . 78

Index **79**

A python package with command-line tools for contextually classifying variable/transient astronomical sources. Sherlock mines a library of historical and on-going astronomical survey data in an attempt to identify sources transient/variable events, and predict their classifications based on the associated crossmatched data.

Here's a summary of what's included in the python package:

Classes

`sherlock.catalogue_conesearch`	*The worker class for the conesearch module*
`sherlock.commonutils.update_wiki_pages`	*Update sherlock's github wiki pages with some useful info regarding the crossmatch database catalogue tables*
`sherlock.database`	*the database object for sherlock, setting up ssh tunnels and various database connections*
`sherlock.database_cleaner`	*Clean and maintain the database helper tables used by sherlock*
`sherlock.imports._base_importer`	*The base importer object used to import new catalgues into sherlock-catalogues database*
`sherlock.imports.ifs`	*Importer for the Multi Unit Spectroscopic Explorer (MUSE) IFS galaxy catalogue stream*
`sherlock.imports.marshall`	*Import the ePESSTO Marshall transient streams (includes multiple on-going transient survey streams) into the Sherlock-catalogues database*
`sherlock.imports.ned`	*Using a list of coordinates, query the online NED database and import sources found within a given search radius of each of the loctions into the sherlock-catalogues database*
`sherlock.imports.ned_d`	*Import the * 'NED-D <https://ned.ipac.caltech.edu/Library/Distances/>'_ *galaxy catalogue in to the sherlock-catalogues database*
`sherlock.imports.veron`	*importer object for the VERON AGN catalogue*
`sherlock.transient_catalogue_crossmatch`	*crossmatch a list of transients against a suite of catalogues according to given search algorithm*
`sherlock.transient_classifier`	*The Sherlock Transient Classifier*

Functions

`sherlock.commonutils.get_crossmatch_catalogues_column_map`	*Query the sherlock-catalogues helper tables to generate a map of the important columns of each catalogue*

CHAPTER 1

Installation

Although you can get Sherlock from a simple `pip` install, it's best to install it within a Conda environment under Anaconda. If you're not familiar with Anaconda, you'll find a good tutorial here to get you up and running.

Once you have Anaconda installed, go ahead and create a new Conda environment to host Sherlock:

```
conda create -n sherlock python=2.7 pip numpy
```

Now activate the environment and install sherlock:

```
source activate sherlock
pip install qub-sherlock
```

At any point in the future you can upgrade to the latest version of sherlock with the command:

```
pip install qub-sherlock --upgrade
```

If instead you want to clone the github repo and install from a local version of the code:

```
git clone git@github.com:thespacedoctor/sherlock.git
cd sherlock
source activate sherlock
python setup.py install
```

1.1 Development

If you want to tinker with the code, then install in development mode. This means you can modify the installed code from whereever you clone the repo to:

```
git clone git@github.com:thespacedoctor/sherlock.git
cd sherlock
source activate sherlock
python setup.py develop
```

Pull requests are welcomed!

1.1.1 Sublime Snippets

If you use Sublime Text as your code editor, and you're planning to develop your own python code with sherlock, you might find my Sublime Snippets useful.

1.2 Issues

Please report any issues here.

CHAPTER 2

Command-Line Usage

```
# SHERLOCK #
: INFERING TRANSIENT-SOURCE CLASSIFICATIONS FROM SPATIALLY CROSS-MATCHED CATALOGUED␣
↪SOURCES :
=======================================================================================

Documentation for sherlock can be found here: http://qub-sherlock.readthedocs.org/en/
↪stable

.. todo ::

    - docuument cl_utils module
    - tidy usage text

Usage:
    sherlock init
    sherlock info [-s <pathToSettingsFile>]
    sherlock [-NA] dbmatch [--update] [-s <pathToSettingsFile>]
    sherlock [-bN] match -- <ra> <dec> [<pathToSettingsFile>]
    sherlock clean [-s <pathToSettingsFile>]
    sherlock wiki [-s <pathToSettingsFile>]
    sherlock import ned <ra> <dec> <radiusArcsec> [-s <pathToSettingsFile>]
    sherlock import cat <cat_name> <pathToDataFile> <cat_version> [-s
↪<pathToSettingsFile>]
    sherlock import stream <stream_name> [-s <pathToSettingsFile>]

Options:
    init                    setup the sherlock settings file for the first time
    match                   XXXX
    dbmatch                 database match
    clean                   XXXX
    wiki                    XXXX
    import                  XXXX
    ned                     use the online NED database as the source catalogue
    cat                     import a static catalogue into the sherlock-catalogues␣
↪database
```

(continues on next page)

(continued from previous page)

```
    stream                  download/stream new data from a give source catalogue␣
↪into the sherlock sherlock-catalogues database
    info                    print an overview of the current catalogues, views and␣
↪streams in the sherlock database ready for crossmatching

    ra                      the right-ascension coordinate with which to perform a␣
↪conesearch (sexegesimal or decimal degrees)
    dec                     the declination coordinate with which to perform a␣
↪conesearch (sexegesimal or decimal degrees)
    radiusArcsec            radius in arcsec of the footprint to download from the␣
↪online NED database
    cat_name                name of the catalogue being imported (veron ned_d)
    stream_name             name of the stream to import into the sherlock-catalogues␣
↪database (ifs)

    -N, --skipNedUpdate     do not update the NED database before classification
    -A, --skipMagUpdate     do not update the peak magnitudes and human readable text␣
↪annotations of objects (can eat up some time)
    -h, --help              show this help message
    -s, --settings          the settings file
    -b, --verbose           print more details to stdout
    -l, --transientlistId   the id of the transient list to classify
    -u, --update            update the transient database with new classifications␣
↪and crossmatches
    -v, --version           print the version of sherlock
```

CHAPTER 3

Documentation

Documentation for sherlock is hosted by Read the Docs (last stable version and latest version).

CHAPTER 4

Sherlock Tutorial

Before you begin using sherlock you will need to populate some custom parameters within the sherlock settings file.

To setup the default settings file at `~/.config/sherlock/sherlock.yaml` run the command:

```
sherlock init
```

This should create and open a new config file; follow the instructions in the file to populate the missing parameters values (usually given an `XXX` placeholder).

Todo:

- add tutorial

4.1 Initialisation and Setup

4.1.1 Populating Sherlock's Settings File

The settings file now contains every option required to change the way the code runs, including database settings and the actual search algorithm.

4.1.1.1 Database Settings

```
database settings:
    static catalogues:
        db: crossmatch_catalogues
        host: 127.0.0.1
        user: pessto
        password: p355t0
```

(continues on next page)

(continued from previous page)

```
    transients:
        user: pessto
        password: p355t0
        db: ps13pipublic
        host: 127.0.0.1
        transient table: tcs_transient_objects
        transient query: "select id as 'id', followup_id as 'alt_id', ra_psf 'ra',␣
↪dec_psf 'dec', local_designation 'name', object_classification as 'object_
↪classification'
            from tcs_transient_objects
            where detection_list_id = 2
            and object_classification is null
            order by followup_id"
        transient id column: id
        transient classification column: object_classification

    pessto marshall:
        user: pessto
        password: p355t0
        db: pessto_marshall
        host: 127.0.0.1
```

The `static catalogues` settings are the settings for connecting to the static catalogues database. Do not edit these settings unless you know what you're doing. If you have your RSA key on starbase, the code will setup a ssh-tunnel for you so that you can connect to this database remotely.

The `transients` settings are for the database you have your transients stored in. `transient table` is the name of the table containing your transients, `transient query` is the SQL query that need executed to get the following info for the transients needing classified:

- `id` - the primary ID for the transient in the database
- `alt_id` - human readable name (optional)
- `ra` - the ra of the object
- `dec` - the dec of the object
- `name` - a further alt id (optional)

The `transient id column` is the primary ID column in the transient database and `transient classification column` is the column you wish to add the classification to.

4.1.1.2 The Search Algorithm

The order searches appear in the `search algorithm` section is the order they shall be run in the actual code:

```
search algorithm:
    sdss qso:
        angular radius arcsec: 2.0
        transient classification: AGN
        database table: tcs_cat_v_sdss_dr9_spect_qsos
    milliquas:
        angular radius arcsec: 3.0
        transient classification: AGN
        database table: tcs_cat_milliquas
    veron:
```

(continues on next page)

(continued from previous page)

```
        angular radius arcsec: 2.0
        transient classification: AGN
        database table: tcs_veron_cat
    ned qso:
        angular radius arcsec: 2.0
        transient classification: AGN
        database table: tcs_cat_v_ned_qsos
    ned nt:
        angular radius arcsec: 3.0
        physical radius kpc: 0.5
        transient classification: NT
        database table: tcs_cat_v_ned_galaxies
    sdss spec nt:
        angular radius arcsec: 3.0
        physical radius kpc: 0.5
        transient classification: NT
        database table: tcs_cat_v_sdss_dr9_spect_galaxies
    sdss phot nt:
        angular radius arcsec: 0.5
        transient classification: NT
        database table: tcs_cat_v_sdss_dr9_galaxies_notspec
    ...
```

The first time you run `sherlock` you will be told to add your settings to the empty settings file that's been created in `yaml`.

For details about all of the catalogue in the catalogues database, run:

```
sherlock info
```

4.2 Classifying Transients

4.2.1 A Single Transient Classification

4.2.2 Classifying Transients in a Transient Database

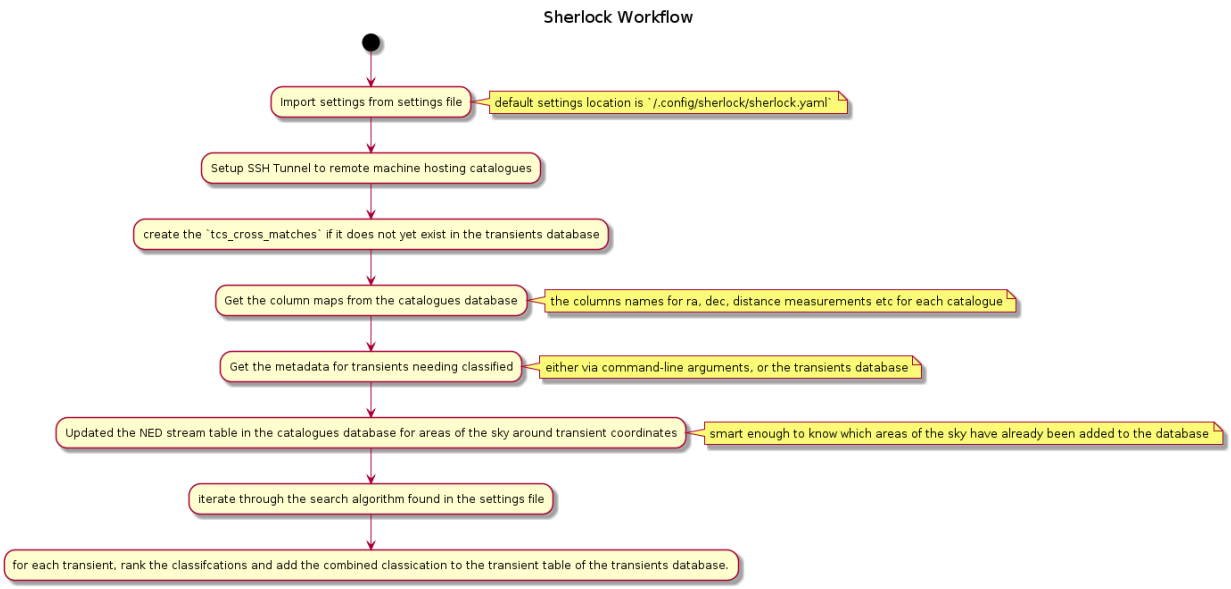
4.2.3 The Classification Workflow

4.2.3.1 Synonyms vs Associations

Sherlock distinguishes between what it views as transient objects synonymous with a catalogued source (the same as or very closely linked to), *synonyms*, and those it deems as merely associated with the catalogued source, *associations*.

Examples of transient-synonym matches are CVs, AGN and variable stars (VS) that match within 1-2 arcsec of their catalogue counterpart. Stretching the definition of *synonym* a little, Sherlock will also match transients close to the centre of galaxies as synonyms[1]. Transient-associations include those transients that are located near, but not on top of, a catalogued source. Example of these associations are 'transients' matching close to bright-stars and are classified as bright-star artefacts (BS) resulting from poor image subtractions near bright stars ($> 14 - 16^{th}$ mag) or transients matched near to a galaxy which may be classified as supernovae (SN). By definition synonyms are a more secure match than associations.

[1] could be classified as a nuclear transient or supernova depending on search algorithm parameters



Each search algorithm module should contain a *synonym* and an *association* key-value sets. For example here is a Guide-Star Catalogue search module:

```
gsc bright stars:
 angular radius arcsec: 100.0
 synonym: VS
 association: BS
 database table: tcs_view_star_guide_star_catalogue_v2_3
 bright mag column: B
 bright limit: 16.
```

If a transient is matched on top of a source in the GSC it's identified as a synonym and classified as a variable star, but if it is match near to the source but not co-located if may been identified as an association and classified as a potential bright-star artefact (BS).

There's also a top-level `synonym radius arcsec` parameter in the Sherlock settings file that defines the maximum transient-catalogue source separation that secures a synonym identification.

```
synonym radius arcsec: 0.5
```

Sherlock performs a two-staged catalogue match, first looking for synonym matches and then for associations. For an individual transient if a synonym match is found within the first search stage the second search stage for associations is skipped as it becomes irrelevant. For example consider the image below (transients marked in red):

figure:: https://farm3.staticflickr.com/2772/33007793206_6dd3e34a21_o.jpg%20title=%22Sherlock%20synonyms%20and%20associations%22%20width=600px

The first stage search should match transients A, C and E as synonyms (NT, VS, VS), these transients are then removed from a further association search. The second stage search then flags B as associated with the large galaxy at the centre of the image and transient D as either associated with the bright-star in the bottom right corner of the image or with the galaxy in the centre.

4.2.3.2 NED Stream Updater

The settings in the settings file relating to the NED stream are:

```
ned stream search radius arcec: 300
first pass ned search radius arcec: 240
ned stream refresh rate in days: 90
```

To update the NED stream, for each transient coordinates the code does a conesearch on the `tcs_helper_ned_query_history` table to see if a search has already been performed within the designated `ned stream refresh rate in days`. If a match isn't found then NED is queried and the `tcs_helper_ned_query_history` is updated for the transient coordinates.

4.2.3.3 Search Algoritm

The algorithm is written and modified within the `sherlock.yaml` settings file. This means you can modify the algorithm without affecting anyone else's search (as long as you are working off the different transient databases).

```
search algorithm:
    sdss qso:
        angular radius arcsec: 2.0
        transient classification: AGN
        database table: tcs_view_qso_sdss_spect_galaxies_qsos_dr12
        stop algorithm on match: False
        match nearest source only: False
    milliquas:
        angular radius arcsec: 3.0
        transient classification: AGN
        database table: tcs_view_agn_milliquas_v4_5
        stop algorithm on match: False
        match nearest source only: False
    veron:
        angular radius arcsec: 2.0
        transient classification: AGN
        database table: tcs_view_agn_veron_v13
        stop algorithm on match: False
        match nearest source only: False
    ned qso:
        angular radius arcsec: 2.0
        transient classification: AGN
        ...
```

Note, to remove a module temporarily, simply comment it out in the settings file (yaml treats lines beginning with # as comments).

Behind the scenes there are 2 types of searches performed on the catalogues.

1. Angular Separation Search
2. Physical Separation Search

4.2.3.4 Angular Separation Search

An example of an angular separation search looks like this in the settings file:

```
milliquas:
    angular radius arcsec: 2.0
    transient classification: AGN
    database table: tcs_view_agn_milliquas_v4_5
```

(continues on next page)

(continued from previous page)

```
    stop algorithm on match  False
    match nearest source only  False
```

The code performs a cone-search on `database table` using the `angular radius arcsec`. If matches are found the associated transient is given a `transient classification` and the results are added to the `tcs_cross_matches` table of the transients database. If `stop algorithm on match` is true the code breaks out of the search algorithm and starts afresh with the next transient to be classified, otherwise the algorithm contines and all matches are recorded in the `tcs_cross_matches` table. If `match nearest source only` is true only the closest match from each catalogue query is be recorded in the `tcs_cross_matches` table.

4.2.3.5 Physical Separation Search

If the `physical radius kpc` key is found in the conesearch module then a physical separation search is performed. First of all an angular cone-search is performed at the coordinates using a suitably large search radius. After this a further search is done on the physical distance parameters returned (distance, physical separation distance, semi-major axis length . . .) for each match.

A physical match is found if:

- The transient falls within 1.5 x semi-major axis of a galaxy
- The transient is within the `physical radius kpc` of a galaxy

As before, all matches are recorded in the `tcs_cross_matches` table.

4.2.3.6 Classification Rankings

If transients are found:

- within 2.0 arc of source, **OR**
- within 20 kpc of host galaxy **AND** assigned a SN classification, **OR**
- within 1.2 times the semi-major axis of the host **AND** assigned a SN classification

they are all given the same top level ranking for classification. After this catalogue weights come into effect to determine the orders of classifications. The catalogue weights are found in the [`tcs_helper_catalogue_tables_info`](Crossmatch Catalogue Tables) table of the catalogues database and give an indication of the accuracy of the classifications of sources in the catalogue. For example the `tcs_cat_sdss_spect_galaxies_qsos_dr12` is given a greater weight than `tcs_cat_sdss_photo_stars_galaxies_dr12` as classifications of the objects based on spectral observations is more accurate than photometry alone.

Once the classifications for each individual transient are ranked, a final, ordered classification listing is given to the transient within its original database table. For example `SN/VARIABLE STAR` means the the transient is most likely a SN but may also be a variable star.

A transient is matched against a source in the sherlock-catalogues because it is either synonymous with a point-like catalogue source (e.g. a variable star or an AGN) or it is hosted by the catalogue source (e.g. supernova, nuclear transient).

A synonymous crossmatch is always a simple angular crossmatch with a search radius that reflects the astrometric accuracy of the RMS combined astrometric errors of the transient source location and that of the catalogue being matched against.

4.3 Sherlock's Catalogue Database

4.3.1 Database Table Naming Scheme

There's a [strict table naming syntax for the crossmatch-catalogues](Crossmatch-Catalogues Database Scheme) database to help deal with catalogue versioning (as updated versions of out sherlock-catalogues are released) and to help ease the burden of modifying crossmatch algorithms employed.

[See here for an up-to-date list of the crossmatch-catalogues](Crossmatch Catalogue Tables) and the [views](Crossmatch Catalogue Views) found on those tables.

4.3.1.1 Table Classes

There are 4 classes of tables in the `crossmatch_catalogues` database:

Table Type | Prefix | Notes | Example |
:———— | :———— | :———— | :———— |
Catalogue | `tcs_cat` | The table is named with the scheme `tcs_cat_` <catalogue name> <version> | `tcs_cat_ned_d_v10_2_0` |
View | `tcs_view` | The view is named with the scheme `tcs_view_` <object type contained> <source table name> | `tcs_view_galaxies_ned_d` |
Helper | `tcs_helper` | Mostly used to store relational information, notes on database tables and book-keeper info | `tcs_helper_catalogue_tables_info` |
Legacy | `legacy_tcs_` | Legacy tables used in previous incarnations of the transient classifier | `legacy_tcs_cat_md01_chiappetti2005` |

4.3.1.2 Versioning

Each catalogue is versioned by appending a version indicator to the end of the table name. There are 3 indicator types:

1. `_final` to show that the catalogue is now at it's final version and shall remain unchanged.
2. `_stream` to show that the catalogue is constantly being updated
3. `_vX_X` to show a version number for the catalogue, e.g. for v10.2 this would be `_v10_2`. We can also have data-release versions (e.g. `_dr12`).

4.3.2 Maintainance and Updates of Catalogues Database

Todo:

- write about marshall stream updates
- write about helper table updates
- write that some tasks need automated

There are various cron-scripts that run on PESSTO-VM03 to automate some tasks. These tasks include

- updating of data-streams into the crossmatch-catalogues database and
- the updates of certain helper tables in the crossmatch-catalogues database.

Currently there are scripts running every:

- 5 mins
- 30 mins
- 1 hr
- 3 hrs
- 12 hrs
- 24 hrs

4.3.3 Updating Catalogues and Adding New Catalogues to the Database

---

Todo:

- list current catalogue importers and how to use them
- add tutorial about creating a new importer
- add steps for adding a catalogue to the database and the search algorithm
- add details about updating the column map
- write code into conf.py to generate tables for docs and link them from here (views, tables and streams)

Using the `sherlock-import` command it's possible to **import and update various catalogues and data-streams** including Milliquas, Veron AGN and the NED-D catalogues. [See here for details](Catalogue Importers).

```
sherlock-importers cat <cat_name> <pathToDataFile> <cat_version> [-s
↪<pathToSettingsFile>]
sherlock-importers stream <stream_name> [-s <pathToSettingsFile>]
```

The command to **import new versions of catalogues** and **data streams** into the `crossmatch_catalogues` database is:

```
Usage:
    sherlock-importers cat <cat_name> <pathToDataFile> <cat_version> [-s
↪<pathToSettingsFile>]
    sherlock-importers stream <stream_name> [-s <pathToSettingsFile>]
```

For example:

```
> sherlock-importers cat milliquas ~/Desktop/milliquas.txt 4.5
1153111 / 1153111 milliquas data added to memory
1153111 / 1153111 rows inserted into tcs_cat_milliquas_v4_5
5694 / 5694 htmIds added to tcs_cat_milliquas_v4_5
```

The command currently supports imports for the following **catalogues**:

- Milliquas
- Veron AGN
- NED-D

Using the command:

```
sherlock-importers stream pessto
```

will import all of the various **data-streams** added to the PESSTO marshall (ASASSN, CRTS, LSQ, PSST . . .).

THE COLUMN MAP LIFTED FROM `tcs_helper_catalogue_tables_info` TABLE IN CATALOGUE DATABASE (COLUMN NAMES ENDDING WITH 'ColName')

CHAPTER 5

Installation

Although you can get Sherlock from a simple `pip` install, it's best to install it within a Conda environment under Anaconda. If you're not familiar with Anaconda, you'll find a good tutorial here to get you up and running.

Once you have Anaconda installed, go ahead and create a new Conda environment to host Sherlock:

```
conda create -n sherlock python=2.7 pip numpy
```

Now activate the environment and install sherlock:

```
source activate sherlock
pip install qub-sherlock
```

At any point in the future you can upgrade to the latest version of sherlock with the command:

```
pip install qub-sherlock --upgrade
```

If instead you want to clone the github repo and install from a local version of the code:

```
git clone git@github.com:thespacedoctor/sherlock.git
cd sherlock
source activate sherlock
python setup.py install
```

5.1 Development

If you want to tinker with the code, then install in development mode. This means you can modify the installed code from whereever you clone the repo to:

```
git clone git@github.com:thespacedoctor/sherlock.git
cd sherlock
source activate sherlock
python setup.py develop
```

Pull requests are welcomed!

5.1.1 Sublime Snippets

If you use Sublime Text as your code editor, and you're planning to develop your own python code with sherlock, you might find my Sublime Snippets useful.

5.2 Issues

Please report any issues here.

CHAPTER 6

Command-Line Usage

```
# SHERLOCK #
: INFERING TRANSIENT-SOURCE CLASSIFICATIONS FROM SPATIALLY CROSS-MATCHED CATALOGUED␣
↪SOURCES :
=======================================================================================

Documentation for sherlock can be found here: http://qub-sherlock.readthedocs.org/en/
↪stable

.. todo ::

    - docuument cl_utils module
    - tidy usage text

Usage:
    sherlock init
    sherlock info [-s <pathToSettingsFile>]
    sherlock [-NA] dbmatch [--update] [-s <pathToSettingsFile>]
    sherlock [-bN] match -- <ra> <dec> [<pathToSettingsFile>]
    sherlock clean [-s <pathToSettingsFile>]
    sherlock wiki [-s <pathToSettingsFile>]
    sherlock import ned <ra> <dec> <radiusArcsec> [-s <pathToSettingsFile>]
    sherlock import cat <cat_name> <pathToDataFile> <cat_version> [-s
↪<pathToSettingsFile>]
    sherlock import stream <stream_name> [-s <pathToSettingsFile>]

Options:
    init                    setup the sherlock settings file for the first time
    match                   XXXX
    dbmatch                 database match
    clean                   XXXX
    wiki                    XXXX
    import                  XXXX
    ned                     use the online NED database as the source catalogue
    cat                     import a static catalogue into the sherlock-catalogues␣
↪database
```

(continues on next page)

(continued from previous page)

```
    stream                  download/stream new data from a give source catalogue␣
↪into the sherlock sherlock-catalogues database
    info                    print an overview of the current catalogues, views and␣
↪streams in the sherlock database ready for crossmatching

    ra                      the right-ascension coordinate with which to perform a␣
↪conesearch (sexegesimal or decimal degrees)
    dec                     the declination coordinate with which to perform a␣
↪conesearch (sexegesimal or decimal degrees)
    radiusArcsec            radius in arcsec of the footprint to download from the␣
↪online NED database
    cat_name                name of the catalogue being imported (veron ned_d)
    stream_name             name of the stream to import into the sherlock-catalogues␣
↪database (ifs)

    -N, --skipNedUpdate     do not update the NED database before classification
    -A, --skipMagUpdate     do not update the peak magnitudes and human readable text␣
↪annotations of objects (can eat up some time)
    -h, --help              show this help message
    -s, --settings          the settings file
    -b, --verbose           print more details to stdout
    -l, --transientlistId   the id of the transient list to classify
    -u, --update            update the transient database with new classifications␣
↪and crossmatches
    -v, --version           print the version of sherlock
```

CHAPTER 7

Documentation

Documentation for sherlock is hosted by Read the Docs (last stable version and latest version).

CHAPTER 8

Sherlock Tutorial

Before you begin using sherlock you will need to populate some custom parameters within the sherlock settings file.

To setup the default settings file at `~/.config/sherlock/sherlock.yaml` run the command:

```
sherlock init
```

This should create and open a new config file; follow the instructions in the file to populate the missing parameters values (usually given an `XXX` placeholder).

Todo:

- add tutorial

8.1 Initialisation and Setup

8.1.1 Populating Sherlock's Settings File

The settings file now contains every option required to change the way the code runs, including database settings and the actual search algorithm.

8.1.1.1 Database Settings

```
database settings:
    static catalogues:
        db: crossmatch_catalogues
        host: 127.0.0.1
        user: pessto
        password: p355t0
```

(continues on next page)

(continued from previous page)

```
    transients:
        user: pessto
        password: p355t0
        db: ps13pipublic
        host: 127.0.0.1
        transient table: tcs_transient_objects
        transient query: "select id as 'id', followup_id as 'alt_id', ra_psf 'ra',␣
↪dec_psf 'dec', local_designation 'name', object_classification as 'object_
↪classification'
            from tcs_transient_objects
            where detection_list_id = 2
            and object_classification is null
            order by followup_id"
        transient id column: id
        transient classification column: object_classification

    pessto marshall:
        user: pessto
        password: p355t0
        db: pessto_marshall
        host: 127.0.0.1
```

The `static catalogues` settings are the settings for connecting to the static catalogues database. Do not edit these settings unless you know what you're doing. If you have your RSA key on starbase, the code will setup a ssh-tunnel for you so that you can connect to this database remotely.

The `transients` settings are for the database you have your transients stored in. `transient table` is the name of the table containing your transients, `transient query` is the SQL query that need executed to get the following info for the transients needing classified:

- `id` - the primary ID for the transient in the database
- `alt_id` - human readable name (optional)
- `ra` - the ra of the object
- `dec` - the dec of the object
- `name` - a further alt id (optional)

The `transient id column` is the primary ID column in the transient database and `transient classification column` is the column you wish to add the classification to.

8.1.1.2 The Search Algorithm

The order searches appear in the `search algorithm` section is the order they shall be run in the actual code:

```
search algorithm:
    sdss qso:
        angular radius arcsec: 2.0
        transient classification: AGN
        database table: tcs_cat_v_sdss_dr9_spect_qsos
    milliquas:
        angular radius arcsec: 3.0
        transient classification: AGN
        database table: tcs_cat_milliquas
    veron:
```

(continues on next page)

(continued from previous page)

```
        angular radius arcsec: 2.0
        transient classification: AGN
        database table: tcs_veron_cat
    ned qso:
        angular radius arcsec: 2.0
        transient classification: AGN
        database table: tcs_cat_v_ned_qsos
    ned nt:
        angular radius arcsec: 3.0
        physical radius kpc: 0.5
        transient classification: NT
        database table: tcs_cat_v_ned_galaxies
    sdss spec nt:
        angular radius arcsec: 3.0
        physical radius kpc: 0.5
        transient classification: NT
        database table: tcs_cat_v_sdss_dr9_spect_galaxies
    sdss phot nt:
        angular radius arcsec: 0.5
        transient classification: NT
        database table: tcs_cat_v_sdss_dr9_galaxies_notspec
    ...
```

The first time you run `sherlock` you will be told to add your settings to the empty settings file that's been created in `yaml`.

For details about all of the catalogue in the catalogues database, run:

```
sherlock info
```

8.2 Classifying Transients

8.2.1 A Single Transient Classification

8.2.2 Classifying Transients in a Transient Database

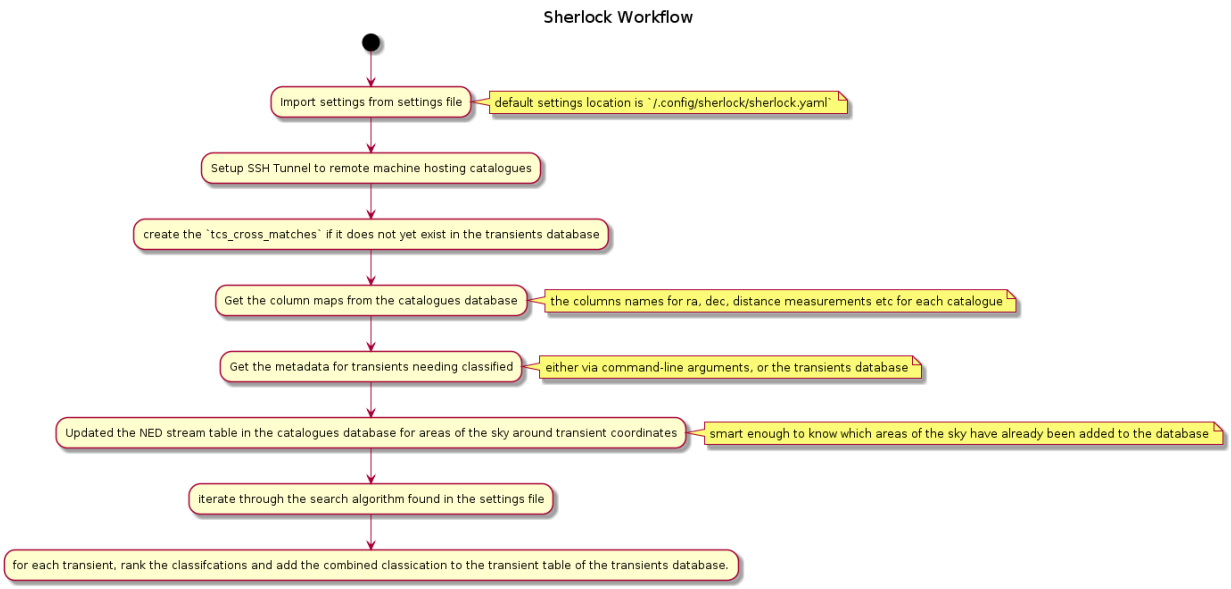
8.2.3 The Classification Workflow

8.2.3.1 Synonyms vs Associations

Sherlock distinguishes between what it views as transient objects synonymous with a catalogued source (the same as or very closely linked to), *synonyms*, and those it deems as merely associated with the catalogued source, *associations*.

Examples of transient-synonym matches are CVs, AGN and variable stars (VS) that match within 1-2 arcsec of their catalogue counterpart. Stretching the definition of *synonym* a little, Sherlock will also match transients close to the centre of galaxies as synonyms[1]. Transient-associations include those transients that are located near, but not on top of, a catalogued source. Example of these associations are 'transients' matching close to bright-stars and are classified as bright-star artefacts (BS) resulting from poor image subtractions near bright stars ($> 14 - 16^{th}$ mag) or transients matched near to a galaxy which may be classified as supernovae (SN). By definition synonyms are a more secure match than associations.

[1] could be classified as a nuclear transient or supernova depending on search algorithm parameters



Each search algorithm module should contain a *synonym* and an *association* key-value sets. For example here is a Guide-Star Catalogue search module:

```
gsc bright stars:
 angular radius arcsec: 100.0
 synonym: VS
 association: BS
 database table: tcs_view_star_guide_star_catalogue_v2_3
 bright mag column: B
 bright limit: 16.
```

If a transient is matched on top of a source in the GSC it's identified as a synonym and classified as a variable star, but if it is match near to the source but not co-located if may been identified as an association and classified as a potential bright-star artefact (BS).

There's also a top-level `synonym radius arcsec` parameter in the Sherlock settings file that defines the maximum transient-catalogue source separation that secures a synonym identification.

```
synonym radius arcsec: 0.5
```

Sherlock performs a two-staged catalogue match, first looking for synonym matches and then for associations. For an individual transient if a synonym match is found within the first search stage the second search stage for associations is skipped as it becomes irrelevant. For example consider the image below (transients marked in red):

figure:: https://farm3.staticflickr.com/2772/33007793206_6dd3e34a21_o.jpg%20title=%22Sherlock%20synonyms%20and%20associations%22%20width=600px

The first stage search should match transients A, C and E as synonyms (NT, VS, VS), these transients are then removed from a further association search. The second stage search then flags B as associated with the large galaxy at the centre of the image and transient D as either associated with the bright-star in the bottom right corner of the image or with the galaxy in the centre.

8.2.3.2 NED Stream Updater

The settings in the settings file relating to the NED stream are:

```
ned stream search radius arcec: 300
first pass ned search radius arcec: 240
ned stream refresh rate in days: 90
```

To update the NED stream, for each transient coordinates the code does a conesearch on the `tcs_helper_ned_query_history` table to see if a search has already been performed within the designated `ned stream refresh rate in days`. If a match isn't found then NED is queried and the `tcs_helper_ned_query_history` is updated for the transient coordinates.

8.2.3.3 Search Algoritm

The algorithm is written and modified within the `sherlock.yaml` settings file. This means you can modify the algorithm without affecting anyone else's search (as long as you are working off the different transient databases).

```
search algorithm:
    sdss qso:
        angular radius arcsec: 2.0
        transient classification: AGN
        database table: tcs_view_qso_sdss_spect_galaxies_qsos_dr12
        stop algorithm on match: False
        match nearest source only: False
    milliquas:
        angular radius arcsec: 3.0
        transient classification: AGN
        database table: tcs_view_agn_milliquas_v4_5
        stop algorithm on match: False
        match nearest source only: False
    veron:
        angular radius arcsec: 2.0
        transient classification: AGN
        database table: tcs_view_agn_veron_v13
        stop algorithm on match: False
        match nearest source only: False
    ned qso:
        angular radius arcsec: 2.0
        transient classification: AGN
        ...
```

Note, to remove a module temporarily, simply comment it out in the settings file (yaml treats lines beginning with # as comments).

Behind the scenes there are 2 types of searches performed on the catalogues.

1. Angular Separation Search
2. Physical Separation Search

8.2.3.4 Angular Separation Search

An example of an angular separation search looks like this in the settings file:

```
milliquas:
    angular radius arcsec: 2.0
    transient classification: AGN
    database table: tcs_view_agn_milliquas_v4_5
```

(continues on next page)

(continued from previous page)

```
    stop algorithm on match  False
    match nearest source only  False
```

The code performs a cone-search on `database table` using the `angular radius arcsec`. If matches are found the associated transient is given a `transient classification` and the results are added to the `tcs_cross_matches` table of the transients database. If `stop algorithm on match` is true the code breaks out of the search algorithm and starts afresh with the next transient to be classified, otherwise the algorithm contines and all matches are recorded in the `tcs_cross_matches` table. If `match nearest source only` is true only the closest match from each catalogue query is be recorded in the `tcs_cross_matches` table.

8.2.3.5 Physical Separation Search

If the `physical radius kpc` key is found in the conesearch module then a physical separation search is performed. First of all an angular cone-search is performed at the coordinates using a suitably large search radius. After this a further search is done on the physical distance parameters returned (distance, physical separation distance, semi-major axis length . . .) for each match.

A physical match is found if:

- The transient falls within 1.5 x semi-major axis of a galaxy
- The transient is within the `physical radius kpc` of a galaxy

As before, all matches are recorded in the `tcs_cross_matches` table.

8.2.3.6 Classification Rankings

If transients are found:

- within 2.0 arc of source, **OR**
- within 20 kpc of host galaxy **AND** assigned a SN classification, **OR**
- within 1.2 times the semi-major axis of the host **AND** assigned a SN classification

they are all given the same top level ranking for classification. After this catalogue weights come into effect to determine the orders of classifications. The catalogue weights are found in the [`tcs_helper_catalogue_tables_info`](Crossmatch Catalogue Tables) table of the catalogues database and give an indication of the accuracy of the classifications of sources in the catalogue. For example the `tcs_cat_sdss_spect_galaxies_qsos_dr12` is given a greater weight than `tcs_cat_sdss_photo_stars_galaxies_dr12` as classifications of the objects based on spectral observations is more accurate than photometry alone.

Once the classifications for each individual transient are ranked, a final, ordered classification listing is given to the transient within its original database table. For example `SN/VARIABLE STAR` means the the transient is most likely a SN but may also be a variable star.

A transient is matched against a source in the sherlock-catalogues because it is either synonymous with a point-like catalogue source (e.g. a variable star or an AGN) or it is hosted by the catalogue source (e.g. supernova, nuclear transient).

A synonymous crossmatch is always a simple angular crossmatch with a search radius that reflects the astrometric accuracy of the RMS combined astrometric errors of the transient source location and that of the catalogue being matched against.

8.3 Sherlock's Catalogue Database

8.3.1 Database Table Naming Scheme

There's a [strict table naming syntax for the crossmatch-catalogues](Crossmatch-Catalogues Database Scheme) database to help deal with catalogue versioning (as updated versions of out sherlock-catalogues are released) and to help ease the burden of modifying crossmatch algorithms employed.

[See here for an up-to-date list of the crossmatch-catalogues](Crossmatch Catalogue Tables) and the [views](Crossmatch Catalogue Views) found on those tables.

8.3.1.1 Table Classes

There are 4 classes of tables in the `crossmatch_catalogues` database:

Table Type | Prefix | Notes | Example |
:———— | :———— | :———— | :———— |
Catalogue | `tcs_cat` | The table is named with the scheme `tcs_cat_` <catalogue name> <version> | `tcs_cat_ned_d_v10_2_0` |
View | `tcs_view` | The view is named with the scheme `tcs_view_` <object type contained> <source table name> | `tcs_view_galaxies_ned_d` |
Helper | `tcs_helper` | Mostly used to store relational information, notes on database tables and book-keeper info | `tcs_helper_catalogue_tables_info` |
Legacy | `legacy_tcs_` | Legacy tables used in previous incarnations of the transient classifier | `legacy_tcs_cat_md01_chiappetti2005` |

8.3.1.2 Versioning

Each catalogue is versioned by appending a version indicator to the end of the table name. There are 3 indicator types:

1. `_final` to show that the catalogue is now at it's final version and shall remain unchanged.
2. `_stream` to show that the catalogue is constantly being updated
3. `_vX_X` to show a version number for the catalogue, e.g. for v10.2 this would be `_v10_2`. We can also have data-release versions (e.g. `_dr12`).

8.3.2 Maintainance and Updates of Catalogues Database

Todo:

- write about marshall stream updates
- write about helper table updates
- write that some tasks need automated

There are various cron-scripts that run on PESSTO-VM03 to automate some tasks. These tasks include

- updating of data-streams into the crossmatch-catalogues database and
- the updates of certain helper tables in the crossmatch-catalogues database.

Currently there are scripts running every:

- 5 mins
- 30 mins
- 1 hr
- 3 hrs
- 12 hrs
- 24 hrs

8.3.3 Updating Catalogues and Adding New Catalogues to the Database

---

Todo:

- list current catalogue importers and how to use them
- add tutorial about creating a new importer
- add steps for adding a catalogue to the database and the search algorithm
- add details about updating the column map
- write code into conf.py to generate tables for docs and link them from here (views, tables and streams)

Using the `sherlock-import` command it's possible to **import and update various catalogues and data-streams** including Milliquas, Veron AGN and the NED-D catalogues. [See here for details](Catalogue Importers).

```
sherlock-importers cat <cat_name> <pathToDataFile> <cat_version> [-s
↪<pathToSettingsFile>]
sherlock-importers stream <stream_name> [-s <pathToSettingsFile>]
```

The command to **import new versions of catalogues** and **data streams** into the `crossmatch_catalogues` database is:

```
Usage:
    sherlock-importers cat <cat_name> <pathToDataFile> <cat_version> [-s
↪<pathToSettingsFile>]
    sherlock-importers stream <stream_name> [-s <pathToSettingsFile>]
```

For example:

```
> sherlock-importers cat milliquas ~/Desktop/milliquas.txt 4.5
1153111 / 1153111 milliquas data added to memory
1153111 / 1153111 rows inserted into tcs_cat_milliquas_v4_5
5694 / 5694 htmIds added to tcs_cat_milliquas_v4_5
```

The command currently supports imports for the following **catalogues**:

- Milliquas
- Veron AGN
- NED-D

Using the command:

```
sherlock-importers stream pessto
```

will import all of the various **data-streams** added to the PESSTO marshall (ASASSN, CRTS, LSQ, PSST . . .).

THE COLUMN MAP LIFTED FROM `tcs_helper_catalogue_tables_info` TABLE IN CATALOGUE DATABASE (COLUMN NAMES ENDDING WITH 'ColName')

8.3.3.1 Subpackages

`sherlock.commonutils`	*common tools used throughout package*
`sherlock.imports`	

8.3.3.1.1 sherlock.commonutils (*subpackage*)

common tools used throughout package

Subpackage Members

get_crossmatch_catalogues_column_map(dbConn, log)	*Query the sherlock-catalogues helper tables to generate a map of the important columns of each catalogue*
getpackagepath	*Get common file and folder paths for the host package - used in unit testing code*
update_wiki_pages(log[, settings])	*Update sherlock's github wiki pages with some useful info regarding the crossmatch database catalogue tables*

8.3.3.1.2 sherlock.imports (*subpackage*)

Subpackage Members

ifs(log[, settings, pathToDataFile, . . .])	*Importer for the Multi Unit Spectroscopic Explorer (MUSE) IFS galaxy catalogue stream*
marshall(log[, settings, pathToDataFile, . . .])	*Import the ePESSTO Marshall transient streams (includes multiple on-going transient survey streams) into the Sherlock-catalogues database*
ned(log[, settings, pathToDataFile, . . .])	*Using a list of coordinates, query the online NED database and import sources found within a given search radius of each of the loctions into the sherlock-catalogues database*
ned_d(log[, settings, pathToDataFile, . . .])	*Import the * 'NED-D <https://ned.ipac.caltech.edu/Library/Distances/>'_ *galaxy catalogue in to the sherlock-catalogues database*
veron(log[, settings, pathToDataFile, . . .])	*importer object for the VERON AGN catalogue*
_base_importer(log[, settings, . . .])	*The base importer object used to import new catalgues into sherlock-catalogues database*

8.3.3.2 Classes

sherlock.catalogue_conesearch	*The worker class for the conesearch module*
sherlock.commonutils.update_wiki_pages	*Update sherlock's github wiki pages with some useful info regarding the crossmatch database catalogue tables*
sherlock.database	*the database object for sherlock, setting up ssh tunnels and various database connections*
sherlock.database_cleaner	*Clean and maintain the database helper tables used by sherlock*
sherlock.imports._base_importer	*The base importer object used to import new catalgues into sherlock-catalogues database*
sherlock.imports.ifs	*Importer for the Multi Unit Spectroscopic Explorer (MUSE) IFS galaxy catalogue stream*
sherlock.imports.marshall	*Import the ePESSTO Marshall transient streams (includes multiple on-going transient survey streams) into the Sherlock-catalogues database*
sherlock.imports.ned	*Using a list of coordinates, query the online NED database and import sources found within a given search radius of each of the loctions into the sherlock-catalogues database*
sherlock.imports.ned_d	*Import the * 'NED-D <https://ned.ipac.caltech.edu/Library/Distances/>'_ *galaxy catalogue in to the sherlock-catalogues database*
sherlock.imports.veron	*importer object for the VERON AGN catalogue*
sherlock.transient_catalogue_crossmatch	*crossmatch a list of transients against a suite of catalogues according to given search algorithm*
sherlock.transient_classifier	*The Sherlock Transient Classifier*

8.3.3.2.1 sherlock.catalogue_conesearch (*class*)

class sherlock.**catalogue_conesearch**(*log*, *ra*, *dec*, *tableName*, *radiusArcsec*, *colMaps*, *dbConn=False*, *nearestOnly=False*, *physicalSearch=False*, *upperMagnitudeLimit=False*, *lowerMagnitudeLimit=False*, *magnitudeLimitFilter=False*)

The worker class for the conesearch module

Key Arguments:

- `dbConn` – mysql database connection to the catalogues database
- `log` – logger
- `ra` – ra of transient location (sexegesimal or decimal degrees, J2000, single location or list of locations)
- `dec` – dec of transient location (sexegesimal or decimal degrees, J2000, single location or list of locations)
- `tableName` – the name of the database table to perform the conesearch on
- `radius` – radius of the conesearch to perform (arcsec)
- `colMaps` – maps of the important column names for each table/view in the crossmatch-catalogues database

- `nearestOnly` – return only the nearest object. Default *False*
- `physicalSearch` – is this a physical search, so only return matches with distance information. Default *False*
- `upperMagnitudeLimit` – the upper magnitude limit if a magnitude cut is requird with the conesearch. Default *False*
- `lowerMagnitudeLimit` – the lower magnitude limit if a magnitude cut is requird with the conesearch. Default *False*
- `magnitudeLimitFilter` – the filter to use for the magnitude limit if requird. Default *False*, ("_u"|"_g"|"_r"|"_i"|"_z"|"_y"|"U"|"B"|"V"|"R"|"I"|"Z"|"J"|"H"|"K"|"G")

Usage:

To setup your logger, settings and database connections, please use the `fundamentals` package (see tutorial here).

Todo:

- update the package tutorial if needed

The following examples assume you've connected to the various databases and generated the catalogue column maps in the following menner:

```
# SETUP ALL DATABASE CONNECTIONS
from sherlock import database
db = database(
    log=log,
    settings=settings
)
dbConns, dbVersions = db.connect()
transientsDbConn = dbConns["transients"]
cataloguesDbConn = dbConns["catalogues"]
pmDbConn = dbConns["marshall"]

# GET THE COLUMN MAPS FROM THE CATALOGUE DATABASE
from sherlock.commonutils import get_crossmatch_catalogues_column_map
colMaps = get_crossmatch_catalogues_column_map(
    log=log,
    dbConn=cataloguesDbConn
)
```

To perform a single location conesearch on a catalogue in the database, for example the milliquas AGN catalogue:

```
from sherlock import catalogue_conesearch
cs = catalogue_conesearch(
    log=log,
    ra="23:01:07.99",
    dec="-01:58:04.5",
    radiusArcsec=60.,
    colMaps=colMaps,
    tableName="tcs_view_agn_milliquas_v4_5",
    dbConn=cataloguesDbConn,
    nearestOnly=False,
    physicalSearch=False
```

(continues on next page)

(continued from previous page)

```
)

# catalogueMatches ARE ORDERED BY ANGULAR SEPARATION
indices, catalogueMatches = cs.search()

print catalogueMatches
```

The output of this search is:

```
[{'R': 20.1, 'cmSepArcsec': 0.28015184686564643, 'ra': 345.2832267,
↪'catalogue_object_subtype': u'QR', 'z': 0.777, 'dec': -1.9679629,
↪'catalogue_object_id': u'PKS 2258-022'}]
```

Note `catalogue_conesearch` can accept coordinates in sexegesimal or decimal degrees (J200). It can also accept lists of corrdinates:

```
from sherlock import catalogue_conesearch
cs = catalogue_conesearch(
    log=log,
    ra=["23:01:07.99", 45.36722, 13.875250],
    dec=["-01:58:04.5", 30.45671, -25.26721],
    radiusArcsec=60.,
    colMaps=colMaps,
    tableName="tcs_view_agn_milliquas_v4_5",
    dbConn=cataloguesDbConn,
    nearestOnly=False,
    physicalSearch=False
)
```

When passing a list of transient coordinates the returned `indices` value becomes important as this list identifies which transient is matched with which crossmatch results (and possibly multiple crossmatch results)

```
indices, catalogueMatches = cs.search()
for i, c in zip(indices, catalogueMatches):
    print i, c
```

The output of this search is:

```
0 {'R': 20.1, 'cmSepArcsec': 0.28015184686564643, 'ra': 345.2832267,
↪'catalogue_object_subtype': u'QR', 'z': 0.777, 'dec': -1.9679629,
↪'catalogue_object_id': u'PKS 2258-022'}
2 {'R': 19.2, 'cmSepArcsec': 0.81509715903447644, 'ra': 13.875,
↪'catalogue_object_subtype': u'Q', 'z': 2.7, 'dec': -25.2672223,
↪'catalogue_object_id': u'Q 0053-2532'}
```

Todo:

- update key arguments values and definitions with defaults
- update return values and definitions
- update usage examples and text
- update docstring text
- check sublime snippet exists
- clip any useful text to docs mindmap

- regenerate the docs and check redendering of this docstring

---

Methods

`search()`	*trigger the conesearch*

search()
trigger the conesearch

Return:

- `matchIndies` – the indicies of the input transient sources (syncs with `uniqueMatchDicts`)
- `uniqueMatchDicts` – the crossmatch results

Usage:

See class docstring for usage examples

Todo:

- update key arguments values and definitions with defaults
- update return values and definitions
- update usage examples and text
- update docstring text
- check sublime snippet exists
- clip any useful text to docs mindmap
- regenerate the docs and check redendering of this docstring

---

8.3.3.2.2 sherlock.commonutils.update_wiki_pages (*class*)

class sherlock.commonutils.**update_wiki_pages**(*log*, *settings=False*)
Update sherlock's github wiki pages with some useful info regarding the crossmatch database catalogue tables

Key Arguments:

- `log` – logger
- `settings` – the settings dictionary

Usage:

To trigger an update of sherlock's wiki pages to give an overview of the crossmatch table database tables run the following:

```
from sherlock.commonutils import update_wiki_pages
wiki = update_wiki_pages(
    log=log,
    settings=settings
)
wiki.update()
```

Todo:

- create a new script for updating sherlock wiki with the snippet above, remove wiki command from cl-utils and add stand alone scripts to the sherlock repo (cleans up the usage and docs for sherlock)
- harvest text from wiki pages and then delete them: https://github.com/thespacedoctor/sherlock/wiki

Methods

`update`()	Update wiki pages
`_create_md_tables`(tableData, viewData, ...)	generate markdown format tables from the database query results
`_get_stream_view_infos`([trimmed])	query the sherlock-catalogues database streamed data tables' metadata
`_get_table_infos`([trimmed])	query the sherlock-catalogues database table metadata
`_get_view_infos`([trimmed])	query the sherlock-catalogues database view metadata
`_update_github`()	commit the changes and push them to github
`_write_wiki_pages`()	write the markdown formated content of the database tables' metadata to local wiki pages

update()
Update wiki pages

See class docstring for usage

_get_table_infos(*trimmed=False*)
query the sherlock-catalogues database table metadata

_get_view_infos(*trimmed=False*)
query the sherlock-catalogues database view metadata

_get_stream_view_infos(*trimmed=False*)
query the sherlock-catalogues database streamed data tables' metadata

_create_md_tables(*tableData*, *viewData*, *streamData*)
generate markdown format tables from the database query results

Key Arguments:

- `tableData` – the sherlock-catalogues database table metadata.
- `viewData` – the sherlock-catalogues database view metadata.
- `streamData` – the sherlock-catalogues database streamed data tables' metadata.

Return:

- None

_write_wiki_pages()
write the markdown formated content of the database tables' metadata to local wiki pages

_update_github()
commit the changes and push them to github

8.3.3.2.3 sherlock.database (*class*)

class sherlock.**database**(*log*, *settings=False*)

the database object for sherlock, setting up ssh tunnels and various database connections

The returned dictionary of database connections contain the following databases:

- `transients` – the database hosting the transient source data
- `catalogues` – connection to the database hosting the contextual catalogues the transients are to be crossmatched against
- `marshall` – connection to the PESSTO Marshall database

Key Arguments:

- `log` – logger
- `settings` – the settings dictionary

Return:

- `dbConns` – a dictionary of the database connections required by sherlock

Usage:

To setup the sherlock database connections, run the following:

```
# SETUP ALL DATABASE CONNECTIONS
from sherlock import database
db = database(
    log=log,
    settings=settings
)
dbConns, dbVersions = db.connect()
transientsDbConn = dbConns["transients"]
cataloguesDbConn = dbConns["catalogues"]
pmDbConn = dbConns["marshall"]
```

Todo:

- update key arguments values and definitions with defaults
- update return values and definitions
- update usage examples and text
- update docstring text
- check sublime snippet exists
- clip any useful text to docs mindmap
- regenerate the docs and check redendering of this docstring

---

Methods

connect()	connect to the various databases, the credientals and settings of which are found in the sherlock settings file
_checkServer(address, port)	Check that the TCP Port we've decided to use for tunnelling is available
_setup_tunnel(tunnelParameters)	*setup a ssh tunnel for a database connection to port through*

connect()
connect to the various databases, the credientals and settings of which are found in the sherlock settings file

Return:

- transientsDbConn – the database hosting the transient source data
- cataloguesDbConn – connection to the database hosting the contextual catalogues the transients are to be crossmatched against
- pmDbConn – connection to the PESSTO Marshall database

See the class docstring for usage

Todo:

- update key arguments values and definitions with defaults
- update return values and definitions
- update usage examples and text
- update docstring text
- check sublime snippet exists
- clip any useful text to docs mindmap
- regenerate the docs and check redendering of this docstring

---

_setup_tunnel(*tunnelParameters*)
setup a ssh tunnel for a database connection to port through

Key Arguments:

- tunnelParameters – the tunnel parameters found associated with the database settings

Return:

- sshPort – the port the ssh tunnel is connected via

Todo:

- update key arguments values and definitions with defaults
- update return values and definitions
- update usage examples and text
- update docstring text
- check sublime snippet exists

- clip any useful text to docs mindmap
- regenerate the docs and check redendering of this docstring

---

_checkServer(*address*, *port*)
Check that the TCP Port we've decided to use for tunnelling is available

---

Todo:

- update key arguments values and definitions with defaults
- update return values and definitions
- update usage examples and text
- update docstring text
- check sublime snippet exists
- clip any useful text to docs mindmap
- regenerate the docs and check redendering of this docstring

---

8.3.3.2.4 sherlock.database_cleaner (*class*)

class sherlock.**database_cleaner**(*log*, *settings=False*)
Clean and maintain the database helper tables used by sherlock

The helper tables list row counts for tables and views and provide the column maps that help sherlock know which catalogue columns relate to which parameters (e.g. RA, DEC etc)

Key Arguments:

- `dbConn` – mysql database connection
- `log` – logger
- `settings` – the settings dictionary

Usage:

Todo:

- add an entry in the tutorial to clean database tables

```
from sherlock.database_cleaner import database_cleaner
db = database_cleaner(
    log=log,
    settings=settings
)
db.clean()
```

Todo:

- update key arguments values and definitions with defaults
- update return values and definitions

- update usage examples and text
- update docstring text
- check sublime snippet exists
- clip any useful text to docs mindmap
- regenerate the docs and check redendering of this docstring

---

Methods

`clean`()	*clean up and run some maintance tasks on the crossmatch catalogue helper tables*
`_clean_up_columns`()	clean up columns
`_update_tcs_helper_catalogue_tables_info`()	update tcs helper catalogue tables info with new tables
`_update_tcs_helper_catalogue_views_info`()	update tcs helper catalogue tables info with new tables
`_updated_row_counts_in_tcs_helper_catalogue_tables_info`()	updated row counts in tcs catalogue tables

clean()
clean up and run some maintance tasks on the crossmatch catalogue helper tables

---

Todo:

- update key arguments values and definitions with defaults
- update return values and definitions
- update usage examples and text
- update docstring text
- check sublime snippet exists
- clip any useful text to docs mindmap
- regenerate the docs and check redendering of this docstring

---

_updated_row_counts_in_tcs_helper_catalogue_tables_info()
updated row counts in tcs catalogue tables

Todo:

- update key arguments values and definitions with defaults
- update return values and definitions
- update usage examples and text
- update docstring text
- check sublime snippet exists
- clip any useful text to docs mindmap
- regenerate the docs and check redendering of this docstring

---

_update_tcs_helper_catalogue_tables_info_with_new_tables()
update tcs helper catalogue tables info with new tables

Todo:

- update key arguments values and definitions with defaults
- update return values and definitions
- update usage examples and text
- update docstring text
- check sublime snippet exists
- clip any useful text to docs mindmap
- regenerate the docs and check redendering of this docstring

---

_clean_up_columns()
clean up columns

Todo:

- update key arguments values and definitions with defaults
- update return values and definitions
- update usage examples and text
- update docstring text
- check sublime snippet exists
- clip any useful text to docs mindmap
- regenerate the docs and check redendering of this docstring

---

_update_tcs_helper_catalogue_views_info_with_new_views()
update tcs helper catalogue tables info with new tables

Todo:

- update key arguments values and definitions with defaults
- update return values and definitions
- update usage examples and text
- update docstring text
- check sublime snippet exists
- clip any useful text to docs mindmap
- regenerate the docs and check redendering of this docstring

---

8.3.3.2.5 sherlock.imports._base_importer (*class*)

class sherlock.imports.**_base_importer**(*log*, *settings=False*, *pathToDataFile=False*, *version=False*, *catalogueName=''*, *coordinateList=[]*, *radiusArcsec=False*)

The base importer object used to import new catalgues into sherlock-catalogues database

Key Arguments:

- `log` – logger
- `settings` – the settings dictionary
- `pathToDataFIle` – path to the file containing the data to import
- `version` – version number of the catalogue to be imported (e.g. DR12)
- `catalogueName` – name of the catalogue to be imported
- `coordinateList` – list of coordinates (needed for some streamed tables)
- `radiusArcsec` – the radius in arcsec with which to perform the initial NED conesearch. Default *False*

Usage:

To use this base class to write a new importer, create your class like so:

```
class newImporter(_base_importer):
    ...
```

Methods

add_data_to_database_table(dictList[, ...])	*Import data in the list of dictionaries in the requested database table*
_add_htmids_to_database_table()	*Add HTMIDs to database table once all the data has been imported (HTM Levels 10,13,16)*
_update_database_helper_table()	*Update the sherlock catalogues database helper table with the time-stamp of when this catlogue was last updated*

add_data_to_database_table(*dictList*, *createStatement=False*)

Import data in the list of dictionaries in the requested database table

Also adds HTMIDs and updates the sherlock-catalogue database helper table with the time-stamp of when the imported catlogue was last updated

Key Arguments:

- `dictList` - a list of dictionaries containing all the rows in the catalogue to be imported
- `createStatement` - the table's mysql create statement (used to generate table if it does not yet exist in database). Default *False*

Usage:

```
self.add_data_to_database_table(
    dictList=dictList,
    createStatement=createStatement
)
```

Todo:

- Write a checklist for creating a new sherlock database importer

_add_htmids_to_database_table()
Add HTMIDs to database table once all the data has been imported (HTM Levels 10,13,16)

Usage:

```
self._add_htmids_to_database_table()
```

_update_database_helper_table()
Update the sherlock catalogues database helper table with the time-stamp of when this catlogue was last updated

Usage:

```
self._update_database_helper_table()
```

8.3.3.2.6 sherlock.imports.ifs (*class*)

class sherlock.imports.**ifs**(*log*, *settings=False*, *pathToDataFile=False*, *version=False*, *catalogueName=''*, *coordinateList=[]*, *radiusArcsec=False*)
Bases: sherlock.imports._base_importer._base_importer

Importer for the Multi Unit Spectroscopic Explorer (MUSE) IFS galaxy catalogue stream

Key Arguments:

- log – logger
- settings – the settings dictionary

Usage:

To import the IFS catalogue stream into the sherlock-catalogues database, run the following:

```
from sherlock.imports import IFS
stream = IFS(
    log=log,
    settings=settings
)
stream.ingest()
```

Todo:

- abstract this module out into its own stand alone script
- check sublime snippet exists

Methods

add_data_to_database_table(dictList[, ...])	*Import data in the list of dictionaries in the requested database table*
ingest()	*Import the IFS catalogue into the sherlock-catalogues database*
_add_htmids_to_database_table()	*Add HTMIDs to database table once all the data has been imported (HTM Levels 10,13,16)*
_create_dictionary_of_IFS()	*Generate the list of dictionaries containing all the rows in the IFS stream*
_update_database_helper_table()	*Update the sherlock catalogues database helper table with the time-stamp of when this catlogue was last updated*

ingest()
Import the IFS catalogue into the sherlock-catalogues database

The method first generates a list of python dictionaries from the IFS datafile, imports this list of dictionaries into a database table and then generates the HTMIDs for that table.

Usage:

See class docstring for usage

_create_dictionary_of_IFS()
Generate the list of dictionaries containing all the rows in the IFS stream

Return:

- dictList - a list of dictionaries containing all the rows in the IFS stream

Usage:

```
from sherlock.imports import IFS
stream = IFS(
    log=log,
    settings=settings
)
dictList = stream._create_dictionary_of_IFS()
```

_add_htmids_to_database_table()
Add HTMIDs to database table once all the data has been imported (HTM Levels 10,13,16)

Usage:

```
self._add_htmids_to_database_table()
```

_update_database_helper_table()
Update the sherlock catalogues database helper table with the time-stamp of when this catlogue was last updated

Usage:

```
self._update_database_helper_table()
```

add_data_to_database_table(*dictList*, *createStatement=False*)
Import data in the list of dictionaries in the requested database table

Also adds HTMIDs and updates the sherlock-catalogue database helper table with the time-stamp of when the imported catlogue was last updated

Key Arguments:

- `dictList` - a list of dictionaries containing all the rows in the catalogue to be imported
- `createStatement` - the table's mysql create statement (used to generate table if it does not yet exist in database). Default *False*

Usage:

```
self.add_data_to_database_table(
    dictList=dictList,
    createStatement=createStatement
)
```

Todo:

- Write a checklist for creating a new sherlock database importer

---

8.3.3.2.7 sherlock.imports.marshall (*class*)

class `sherlock.imports.`**marshall**(*log*, *settings=False*, *pathToDataFile=False*, *version=False*, *catalogueName=''*, *coordinateList=[]*, *radiusArcsec=False*)

Bases: `sherlock.imports._base_importer._base_importer`

Import the ePESSTO Marshall transient streams (includes multiple on-going transient survey streams) into the Sherlock-catalogues database

Key Arguments:

- `log` – logger
- `settings` – the settings dictionary

Usage:

To import/update the marshall catalogue streams in the sherlock-catalogues database, run the following:

```
from sherlock.imports import marshall
stream = marshall(
    log=log,
    settings=settings
)
stream.ingest()
```

Todo:

- abstract this module out into its own stand alone script
- check sublime snippet exists

Methods

add_data_to_database_table(dictList[, ...])	*Import data in the list of dictionaries in the requested database table*
ingest()	*Ingest the ePESSTO Marshall transient stream into the catalogues database*
_add_htmids_to_database_table()	*Add HTMIDs to database table once all the data has been imported (HTM Levels 10,13,16)*
_create_dictionary_of_marshall(...)	create a list of dictionaries containing all the rows in the marshall stream
_update_database_helper_table()	*Update the sherlock catalogues database helper table with the time-stamp of when this catlogue was last updated*

ingest()
Ingest the ePESSTO Marshall transient stream into the catalogues database

The method first creates the tables for the various marshall feeder surveys in the sherlock-catalogues database (if they do not yet exist). Then the marshall database is queried for each transient survey and the results imported into the sherlock-catalogues tables,

See the class docstring for usage

Todo:

- convert the directory_script_runner to 'load in file'

_create_dictionary_of_marshall(*marshallQuery*, *marshallTable*)
create a list of dictionaries containing all the rows in the marshall stream

Key Arguments:

- `marshallQuery` – the query used to lift the required data from the marshall database.
- `marshallTable` – the name of the marshall table we are lifting the data from.

Return:

- `dictList` - a list of dictionaries containing all the rows in the marshall stream

_add_htmids_to_database_table()
Add HTMIDs to database table once all the data has been imported (HTM Levels 10,13,16)

Usage:

```
self._add_htmids_to_database_table()
```

_update_database_helper_table()
Update the sherlock catalogues database helper table with the time-stamp of when this catlogue was last updated

Usage:

```
self._update_database_helper_table()
```

add_data_to_database_table(*dictList*, *createStatement=False*)
Import data in the list of dictionaries in the requested database table

Also adds HTMIDs and updates the sherlock-catalogue database helper table with the time-stamp of when the imported catlogue was last updated

Key Arguments:

- `dictList` - a list of dictionaries containing all the rows in the catalogue to be imported
- `createStatement` - the table's mysql create statement (used to generate table if it does not yet exist in database). Default *False*

Usage:

```
self.add_data_to_database_table(
    dictList=dictList,
    createStatement=createStatement
)
```

Todo:

- Write a checklist for creating a new sherlock database importer

---

8.3.3.2.8 sherlock.imports.ned (*class*)

class `sherlock.imports.`**ned**(*log*, *settings=False*, *pathToDataFile=False*, *version=False*, *catalogueName=''*, *coordinateList=[]*, *radiusArcsec=False*)

Bases: `sherlock.imports._base_importer._base_importer`

Using a list of coordinates, query the online NED *database and import sources found within a given search radius of each of the loctions into the sherlock-catalogues database*

The code:

1. Uses the list of transient coordinates and queries NED (conesearch) for the results within the given search radius
2. Creates the `tcs_cat_ned_stream` table if it doesn't exist
3. Adds the resulting matched NED IDs/Names to the `tcs_cat_ned_stream` table
4. Updates the NED query history table
5. Queris NED via NED IDs (object search) for the remaining source metadata to be added to the `tcs_cat_ned_stream` table

Note it's up to the user to filter the input coordinate list by checking whether or not the same area of the sky has been imported into the `tcs_cat_ned_stream` table recently (by checking the `tcs_helper_ned_query_history` table)

Key Arguments:

- `dbConn` – mysql database connection
- `log` – logger
- `settings` – the settings dictionary
- `coordinateList` – list of coordinates (a list of strings with RA and DEC space separated)
- `radiusArcsec` - - the radius in arcsec with which to perform the initial NED conesearch. Default * False*

Usage:

To import the ned catalogue stream, run the following:

```
from sherlock.imports import ned
stream = ned(
    log=log,
    settings=settings,
    coordinateList=["23.12323 -12.34343", "345.43234 45.26789"],
    radiusArcsec=180
)
stream.ingest()
```

Todo:

- test this code is still working after changes
- add option to filter coordinate list via the `tcs_helper_ned_query_history` table
- check sublime snippet exists
- clip any useful text to docs mindmap

Methods

`add_data_to_database_table`(dictList[, ...])	*Import data in the list of dictionaries in the requested database table*
`ingest`()	*Perform conesearches of the online NED database and import the results into a the sherlock-database*
`_add_htmids_to_database_table`()	*Add HTMIDs to database table once all the data has been imported (HTM Levels 10,13,16)*
`_count_ned_sources_in_database_requiring_metadata`()	*Count the sources in the NED table requiring metadata*
`_create_dictionary_of_ned`()	*Create a list of dictionaries containing all the object ids (NED names) in the ned stream*
`_do_ned_namesearch_queries_and_add_results_to_database`()	*Query NED via name search and add results metadata to database*
`_download_ned_source_metadata`()	*Query NED using the names of the NED sources in our local database to retrieve extra metadata*
`_get_ned_sources_needing_metadata`()	*Get the names of 50000 or less NED sources that still require metabase in the database*
`_update_database_helper_table`()	*Update the sherlock catalogues database helper table with the time-stamp of when this catlogue was last updated*
`_update_ned_query_history`()	*Update the database helper table to give details of the ned cone searches performed*

ingest()

Perform conesearches of the online NED database and import the results into a the sherlock-database

The code:

1. uses the list of transient coordinates and queries NED for the results within the given search radius
2. Creates the `tcs_cat_ned_stream` table if it doesn't exist
3. Adds the resulting NED IDs/Names to the `tcs_cat_ned_stream` table

4. Updates the NED query history table
5. Queris NED via NED IDs for the remaining source metadata to be added to the `tcs_cat_ned_stream` table

Usage:

Having setup the NED object with a coordinate list and cone-search radius, run the *`ingest()`* method

```
stream.ingest()
```

Todo:

- check sublime snippet exists
- clip any useful text to docs mindmap
- regenerate the docs and check redendering of this docstring

---

`_create_dictionary_of_ned`()
Create a list of dictionaries containing all the object ids (NED names) in the ned stream

Return:

- `dictList` - a list of dictionaries containing all the object ids (NED names) in the ned stream

Usage:

```
dictList = stream._create_dictionary_of_ned()
```

`_update_ned_query_history`()
Update the database helper table to give details of the ned cone searches performed

Usage:

```
stream._update_ned_query_history()
```

`_download_ned_source_metadata`()
Query NED using the names of the NED sources in our local database to retrieve extra metadata

Usage:

```
stream._download_ned_source_metadata()
```

`_get_ned_sources_needing_metadata`()
Get the names of 50000 or less NED sources that still require metabase in the database

Return:

- `len(self.theseIds)` – the number of NED IDs returned

Usage:

```
numberSources = stream._get_ned_sources_needing_metadata()
```

`_do_ned_namesearch_queries_and_add_resulting_metadata_to_database`(*batchCount*)
Query NED via name searcha and add result metadata to database

Key Arguments:

- `batchCount` - the index number of the batch sent to NED (only needed for printing to STDOUT to give user idea of progress)

Usage:

```
numberSources = stream._do_ned_namesearch_queries_and_add_resulting_
↪metadata_to_database(batchCount=10)
```

_count_ned_sources_in_database_requiring_metadata()
Count the sources in the NED table requiring metadata

Return:

- `self.total`, `self.batches` – total number of galaxies needing metadata & the number of batches required to be sent to NED

Usage:

```
totalRemaining, numberOfBatches = stream._count_ned_sources_in_
↪database_requiring_metadata()
```

_add_htmids_to_database_table()
Add HTMIDs to database table once all the data has been imported (HTM Levels 10,13,16)

Usage:

```
self._add_htmids_to_database_table()
```

_update_database_helper_table()
Update the sherlock catalogues database helper table with the time-stamp of when this catlogue was last updated

Usage:

```
self._update_database_helper_table()
```

add_data_to_database_table(*dictList*, *createStatement=False*)
Import data in the list of dictionaries in the requested database table

Also adds HTMIDs and updates the sherlock-catalogue database helper table with the time-stamp of when the imported catlogue was last updated

Key Arguments:

- `dictList` - a list of dictionaries containing all the rows in the catalogue to be imported
- `createStatement` - the table's mysql create statement (used to generate table if it does not yet exist in database). Default *False*

Usage:

```
self.add_data_to_database_table(
    dictList=dictList,
    createStatement=createStatement
)
```

Todo:

- Write a checklist for creating a new sherlock database importer

---

8.3.3.2.9 sherlock.imports.ned_d (*class*)

class sherlock.imports.**ned_d**(*log*, *settings=False*, *pathToDataFile=False*, *version=False*, *catalogueName=''*, *coordinateList=[]*, *radiusArcsec=False*)

Bases: sherlock.imports._base_importer._base_importer

*Import the * 'NED-D <https://ned.ipac.caltech.edu/Library/Distances/>'_ *galaxy catalogue in to the sherlock-catalogues database*

Key Arguments:

- log – logger
- settings – the settings dictionary
- pathToDataFile – path to the ned_d data file
- version – version of the ned_d catalogue
- catalogueName – the name of the catalogue

Usage:

To import the ned_d catalogue catalogue, run the following:

```
from sherlock.imports import ned_d
catalogue = ned_d(
    log=log,
    settings=settings,
    pathToDataFile="/path/to/ned_d.txt",
    version="1.0",
    catalogueName="ned_d"
)
catalogue.ingest()
```

Todo:

- abstract this module out into its own stand alone script

Methods

add_data_to_database_table(dictList[, ...])	*Import data in the list of dictionaries in the requested database table*
ingest()	Import the ned_d catalogue into the catalogues database
_add_htmids_to_database_table()	*Add HTMIDs to database table once all the data has been imported (HTM Levels 10,13,16)*
_clean_up_columns()	clean up columns of the NED table
_count_galaxies_requiring_metadata()	count galaxies requiring metadata
_create_dictionary_of_ned_d()	create a list of dictionaries containing all the rows in the ned_d catalogue
_get_3000_galaxies_needing_metadata()	get 3000 galaxies needing metadata
_get_metadata_for_galaxies()	get metadata for galaxies
_query_ned_and_add_results_to_database(t...	query ned and add results to database

Continued on next page

Table 13 – continued from previous page

`_update_database_helper_table()`	*Update the sherlock catalogues database helper table with the time-stamp of when this catlogue was last updated*
`_update_sdss_coverage()`	update sdss coverage

`ingest``()`
Import the ned_d catalogue into the catalogues database

The method first generates a list of python dictionaries from the ned_d datafile, imports this list of dictionaries into a database table and then generates the HTMIDs for that table.

Usage:

See class docstring for usage

Todo:

- update docstring text
- check sublime snippet exists
- clip any useful text to docs mindmap
- regenerate the docs and check redendering of this docstring

---

`_create_dictionary_of_ned_d``()`
create a list of dictionaries containing all the rows in the ned_d catalogue

Return:

- `dictList` - a list of dictionaries containing all the rows in the ned_d catalogue

---

Todo:

- update key arguments values and definitions with defaults
- update return values and definitions
- update usage examples and text
- update docstring text
- check sublime snippet exists
- clip any useful text to docs mindmap
- regenerate the docs and check redendering of this docstring

---

`_clean_up_columns``()`
clean up columns of the NED table

Todo:

- update key arguments values and definitions with defaults
- update return values and definitions
- update usage examples and text
- update docstring text

- check sublime snippet exists
- clip any useful text to docs mindmap
- regenerate the docs and check redendering of this docstring

---

_get_metadata_for_galaxies()
get metadata for galaxies

Todo:

- update key arguments values and definitions with defaults
- update return values and definitions
- update usage examples and text
- update docstring text
- check sublime snippet exists
- clip any useful text to docs mindmap
- regenerate the docs and check redendering of this docstring

---

_count_galaxies_requiring_metadata()
count galaxies requiring metadata

Return:

- `self.total`, `self.batches` – total number of galaxies needing metadata & the number of batches required to be sent to NED

Todo:

- update key arguments values and definitions with defaults
- update return values and definitions
- update usage examples and text
- update docstring text
- check sublime snippet exists
- clip any useful text to docs mindmap
- regenerate the docs and check redendering of this docstring

---

_get_3000_galaxies_needing_metadata()
get 3000 galaxies needing metadata

Return:

- `len(self.theseIds)` – the number of NED IDs returned

Todo:

- update key arguments values and definitions with defaults
- update return values and definitions

- update usage examples and text
- update docstring text
- check sublime snippet exists
- clip any useful text to docs mindmap
- regenerate the docs and check redendering of this docstring

---

_query_ned_and_add_results_to_database(*batchCount*)
query ned and add results to database

Key Arguments:

- `batchCount` - the index number of the batch sent to NED

Todo:

- update key arguments values and definitions with defaults
- update return values and definitions
- update usage examples and text
- update docstring text
- check sublime snippet exists
- clip any useful text to docs mindmap
- regenerate the docs and check redendering of this docstring

---

_update_sdss_coverage()
update sdss coverage

Todo:

- update key arguments values and definitions with defaults
- update return values and definitions
- update usage examples and text
- update docstring text
- check sublime snippet exists
- clip any useful text to docs mindmap
- regenerate the docs and check redendering of this docstring

---

_add_htmids_to_database_table()
Add HTMIDs to database table once all the data has been imported (HTM Levels 10,13,16)

Usage:

```
self._add_htmids_to_database_table()
```

_update_database_helper_table()
Update the sherlock catalogues database helper table with the time-stamp of when this catlogue was last updated

Usage:

```
self._update_database_helper_table()
```

add_data_to_database_table(*dictList*, *createStatement=False*)
Import data in the list of dictionaries in the requested database table

Also adds HTMIDs and updates the sherlock-catalogue database helper table with the time-stamp of when the imported catlogue was last updated

Key Arguments:

- `dictList` - a list of dictionaries containing all the rows in the catalogue to be imported
- `createStatement` - the table's mysql create statement (used to generate table if it does not yet exist in database). Default *False*

Usage:

```
self.add_data_to_database_table(
    dictList=dictList,
    createStatement=createStatement
)
```

Todo:

- Write a checklist for creating a new sherlock database importer

---

8.3.3.2.10 sherlock.imports.veron (*class*)

class sherlock.imports.**veron**(*log*, *settings=False*, *pathToDataFile=False*, *version=False*, *catalogueName=''*, *coordinateList=[]*, *radiusArcsec=False*)
Bases: `sherlock.imports._base_importer._base_importer`

importer object for the VERON AGN catalogue

Key Arguments:

- `dbConn` – mysql database connection
- `log` – logger
- `settings` – the settings dictionary
- `pathToDataFIle` – path to the veron data file
- `version` – version of the veron catalogue

Usage:

To import the veron catalogue catalogue, run the following:

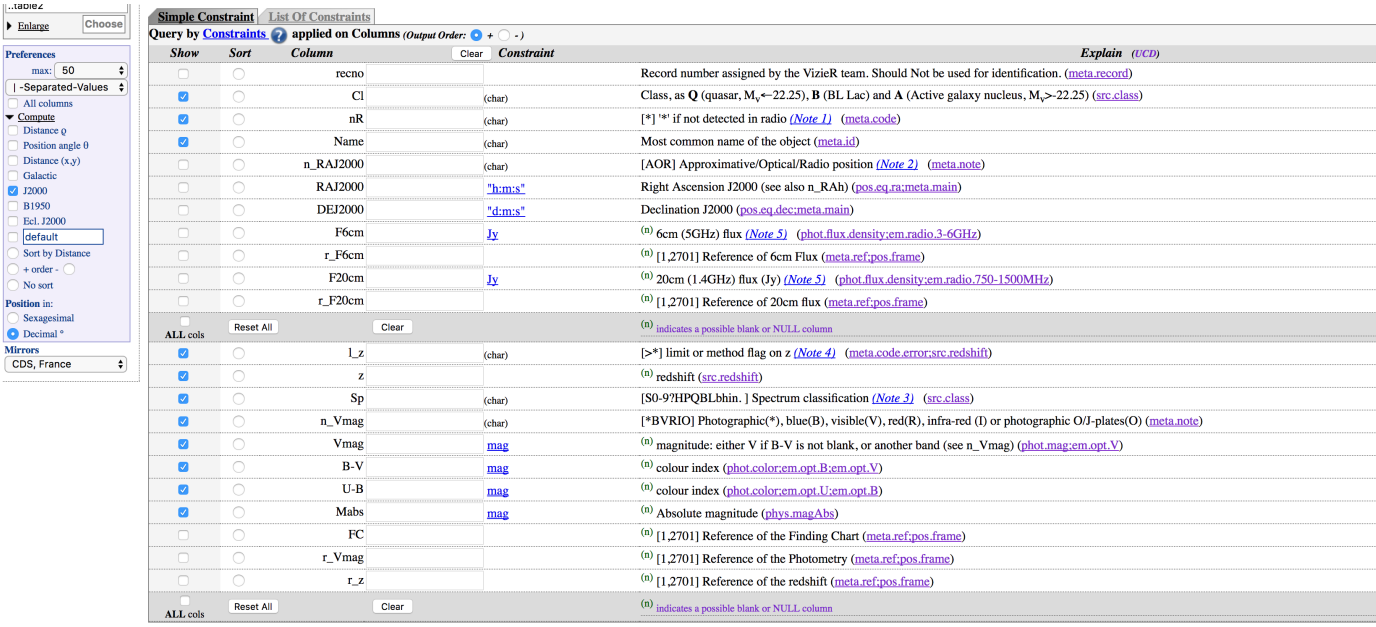
```
from sherlock.imports import veron
catalogue = veron(
    log=log,
    settings=settings,
    pathToDataFile="/path/to/veron.txt",
    version="1.0",
    catalogueName="veron"
```

(continues on next page)

(continued from previous page)

```
)
catalogue.ingest()
```

Whenever downloading a version of the Veron catalogue from Vizier use the following column selection:



Simple Constraint | List Of Constraints

Query by Constraints applied on Columns (Output Order: + -)

Show	Sort	Column	Clear	Constraint	Explain (UCD)
☐	○	recno			Record number assigned by the VizieR team. Should Not be used for identification. (meta.record)
☑	○	Cl		(char)	Class, as **Q** (quasar, M_v<-22.25), **B** (BL Lac) and **A** (Active galaxy nucleus, M_v>-22.25) (src.class)
☑	○	nR		(char)	[*] '*' if not detected in radio *(Note 1)* (meta.code)
☑	○	Name		(char)	Most common name of the object (meta.id)
☐	○	n_RAJ2000		(char)	[AOR] Approximative/Optical/Radio position *(Note 2)* (meta.note)
☐	○	RAJ2000		"h:m:s"	Right Ascension J2000 (see also n_RAh) (pos.eq.ra;meta.main)
☐	○	DEJ2000		"d:m:s"	Declination J2000 (pos.eq.dec;meta.main)
☐	○	F6cm		Jy	(n) 6cm (5GHz) flux *(Note 5)* (phot.flux.density;em.radio.3-6GHz)
☐	○	r_F6cm			(n) [1,2701] Reference of 6cm Flux (meta.ref;pos.frame)
☐	○	F20cm		Jy	(n) 20cm (1.4GHz) flux (Jy) *(Note 5)* (phot.flux.density;em.radio.750-1500MHz)
☐	○	r_F20cm			(n) [1,2701] Reference of 20cm flux (meta.ref;pos.frame)
ALL cols	Reset All		Clear		(n) indicates a possible blank or NULL column
☑	○	l_z		(char)	[>*] limit or method flag on z *(Note 4)* (meta.code.error;src.redshift)
☑	○	z			(n) redshift (src.redshift)
☑	○	Sp		(char)	[S0-9?HPQBLbhin.] Spectrum classification *(Note 3)* (src.class)
☑	○	n_Vmag		(char)	[*BVRIO] Photographic(*), blue(B), visible(V), red(R), infra-red (I) or photographic O/J-plates(O) (meta.note)
☑	○	Vmag		mag	(n) magnitude: either V if B-V is not blank, or another band (see n_Vmag) (phot.mag;em.opt.V)
☑	○	B-V		mag	(n) colour index (phot.color;em.opt.B;em.opt.V)
☑	○	U-B		mag	(n) colour index (phot.color;em.opt.U;em.opt.B)
☑	○	Mabs		mag	(n) Absolute magnitude (phys.magAbs)
☐	○	FC			(n) [1,2701] Reference of the Finding Chart (meta.ref;pos.frame)
☐	○	r_Vmag			(n) [1,2701] Reference of the Photometry (meta.ref;pos.frame)
☐	○	r_z			(n) [1,2701] Reference of the redshift (meta.ref;pos.frame)
ALL cols	Reset All		Clear		(n) indicates a possible blank or NULL column

---

Todo:

- abstract this module out into its own stand alone script
- add ppmxl to stand-alone import scripts
- update key arguments values and definitions with defaults
- update return values and definitions
- update usage examples and text
- update docstring text
- check sublime snippet exists
- clip any useful text to docs mindmap
- regenerate the docs and check redendering of this docstring

---

Methods

`add_data_to_database_table`(dictList[, ...])	*Import data in the list of dictionaries in the requested database table*
`ingest`()	ingest the veron catalogue into the catalogues database
`_add_htmids_to_database_table`()	*Add HTMIDs to database table once all the data has been imported (HTM Levels 10,13,16)*

Continued on next page

Table 14 – continued from previous page

`_create_dictionary_of_veron()`	create a list of dictionaries containing all the rows in the veron catalogue
`_update_database_helper_table()`	*Update the sherlock catalogues database helper table with the time-stamp of when this catlogue was last updated*

`ingest`()
ingest the veron catalogue into the catalogues database

See class docstring for usage.

`_create_dictionary_of_veron`()
create a list of dictionaries containing all the rows in the veron catalogue

Return:

- `dictList` - a list of dictionaries containing all the rows in the veron catalogue

---

Todo:

- update key arguments values and definitions with defaults
- update return values and definitions
- update usage examples and text
- update docstring text
- check sublime snippet exists
- clip any useful text to docs mindmap
- regenerate the docs and check redendering of this docstring

---

`_add_htmids_to_database_table`()
Add HTMIDs to database table once all the data has been imported (HTM Levels 10,13,16)

Usage:

```
self._add_htmids_to_database_table()
```

`_update_database_helper_table`()
Update the sherlock catalogues database helper table with the time-stamp of when this catlogue was last updated

Usage:

```
self._update_database_helper_table()
```

`add_data_to_database_table`(*dictList*, *createStatement=False*)
Import data in the list of dictionaries in the requested database table

Also adds HTMIDs and updates the sherlock-catalogue database helper table with the time-stamp of when the imported catlogue was last updated

Key Arguments:

- `dictList` - a list of dictionaries containing all the rows in the catalogue to be imported
- `createStatement` - the table's mysql create statement (used to generate table if it does not yet exist in database). Default *False*

Usage:

```
self.add_data_to_database_table(
    dictList=dictList,
    createStatement=createStatement
)
```

Todo:

- Write a checklist for creating a new sherlock database importer

8.3.3.2.11 sherlock.transient_catalogue_crossmatch (*class*)

class sherlock.**transient_catalogue_crossmatch**(*log*, *dbConn=False*, *settings=False*, *colMaps=False*, *transients=[]*)

crossmatch a list of transients against a suite of catalogues according to given search algorithm

Key Arguments:

- `dbConn` – mysql database connection for the catalogues
- `log` – logger
- `settings` – the settings dictionary
- `colMaps` – maps of the important column names for each table/view in the crossmatch-catalogues database
- `transients` – the list of transients

Usage:

To setup your logger, settings and database connections, please use the `fundamentals` package (see tutorial here).

To initiate a transient_catalogue_crossmatch object, use the following:

```
from sherlock import transient_catalogue_crossmatch
xmatcher = transient_catalogue_crossmatch(
    log=log,
    settings=settings,
    dbConn=dbConn,
    colMaps=colMaps,
    transients=transients
)
```

Then to run the transient through the search algorithm found in the settings file, use the `match` method:

```
classifications = xmatcher.match()
```

Todo:

- update key arguments values and definitions with defaults
- update return values and definitions
- update usage examples and text

- update docstring text
- check sublime snippet exists
- clip any useful text to docs mindmap
- regenerate the docs and check redendering of this docstring

---

Methods

`angular_crossmatch_against_catalogue`(objectList)	*perform an angular separation crossmatch against a given catalogue in the database and annotate the crossmatch with some value added parameters (distances, physical separations, sub-type of transient etc)*
`match`()	*match the transients against the sherlock-catalogues according to the search algorithm and return matches alongside the predicted classification(s)*
`physical_separation_crossmatch_against_catalogue`(objectList)	*perform a physical separation crossmatch against a given catalogue in the database*
`_annotate_crossmatch_with_value_added_parameters`(crossmatches)	*annotate each crossmatch with physical parameters such are distances etc*
`_bright_star_match`(matchedObjects, ...)	*perform a bright star match on the crossmatch results if required by the catalogue search*
`_galaxy_association_cuts`(matchedObjects, ...)	*perform a bright star match on the crossmatch results if required by the catalogue search*

match()

match the transients against the sherlock-catalogues according to the search algorithm and return matches alongside the predicted classification(s)

Return:

- `classification` – the crossmatch results and classifications assigned to the transients

See the class docstring for usage.

Todo:

- update key arguments values and definitions with defaults
- update return values and definitions
- update usage examples and text
- update docstring text
- check sublime snippet exists
- clip any useful text to docs mindmap
- regenerate the docs and check redendering of this docstring

---

angular_crossmatch_against_catalogue(*objectList*, *searchPara={}*, *search_name=''*, *brightnessFilter=False*, *physicalSearch=False*, *classificationType=False*)

perform an angular separation crossmatch against a given catalogue in the database and annotate the crossmatch with some value added parameters (distances, physical separations, sub-type of transient etc)

Key Arguments:

- `objectList` – the list of transient locations to match against the crossmatch catalogue
- `searchPara` – the search parameters for this individual search as lifted from the search algorithm in the sherlock settings file
- `search_name` – the name of the search as given in the sherlock settings file
- `brightnessFilter` – is this search to be constrained by magnitude of the catalogue sources? Default *False*. [bright|faint|general]
- `physicalSearch` – is this angular search a sub-part of a physical separation search
- `classificationType` – synonym, association or annotation. Default *False*

Return:

- matchedObjects – any sources matched against the object

Usage:

Take a list of transients from somewhere

```
transients = [
    {'ps1_designation': u'PS1-14aef',
     'name': u'4L3Piiq',
     'detection_list_id': 2,
     'local_comments': u'',
     'ra': 0.02548233704918263,
     'followup_id': 2065412L,
     'dec': -4.284933417540423,
     'id': 1000006110041705700L,
     'object_classification': 0L
    },

    {'ps1_designation': u'PS1-13dcr',
     'name': u'3I3Phzx',
     'detection_list_id': 2,
     'local_comments': u'',
     'ra': 4.754236999477372,
     'followup_id': 1140386L,
     'dec': 28.276703631398625,
     'id': 1001901011281636100L,
     'object_classification': 0L
    },

    {'ps1_designation': u'PS1-13dhc',
     'name': u'3I3Pixd',
     'detection_list_id': 2,
     'local_comments': u'',
     'ra': 1.3324973428505413,
     'followup_id': 1202386L,
     'dec': 32.98869220595689,
     'id': 1000519791325919200L,
     'object_classification': 0L
    }
]
```

Then run the `angular_crossmatch_against_catalogue` method to crossmatch against the catalogues and return results:

```
# ANGULAR CONESEARCH ON CATALOGUE
search_name = "ned_d spec sn"
searchPara = self.settings["search algorithm"][search_name]
matchedObjects = xmatcher.angular_crossmatch_against_catalogue(
    objectList=transients,
    searchPara=searchPara,
    search_name=search_name
)
```

Todo:

- update key arguments values and definitions with defaults
- update return values and definitions
- update usage examples and text
- update docstring text
- check sublime snippet exists
- clip any useful text to docs mindmap
- regenerate the docs and check redendering of this docstring

---

_annotate_crossmatch_with_value_added_parameters(*crossmatchDict*, *catalogueName*, *searchPara*, *search_name*)

annotate each crossmatch with physical parameters such are distances etc

Key Arguments:

- `crossmatchDict` – the crossmatch dictionary
- `catalogueName` – the name of the catalogue the crossmatch results from
- `searchPara` – the search parameters for this individual search as lifted from the search algorithm in the sherlock settings file
- `search_name` – the name of the search as given in the sherlock settings file

Return:

- `crossmatchDict` – the annotated crossmatch dictionary

Todo:

- update key arguments values and definitions with defaults
- update return values and definitions
- update usage examples and text
- update docstring text
- check sublime snippet exists
- clip any useful text to docs mindmap
- regenerate the docs and check redendering of this docstring

---

_bright_star_match(*matchedObjects*, *catalogueName*, *magnitudeLimitFilter*, *lowerMagnitudeLimit*)
perform a bright star match on the crossmatch results if required by the catalogue search

Key Arguments:

- `matchedObjects` – the list of matched sources from the catalogue crossmatch
- `catalogueName` – the name of the catalogue the crossmatch results from
- `magnitudeLimitFilter` – the name of the column containing the magnitude to filter on
- `lowerMagnitudeLimit` – the lower magnitude limit to match bright stars against

Return:

- `brightStarMatches` – the trimmed matched sources (bright stars associations only)

Todo:

- update key arguments values and definitions with defaults
- update return values and definitions
- update usage examples and text
- update docstring text
- check sublime snippet exists
- clip any useful text to docs mindmap
- regenerate the docs and check redendering of this docstring

---

_galaxy_association_cuts(*matchedObjects*, *catalogueName*, *magnitudeLimitFilter*, *upperMagnitudeLimit*, *lowerMagnitudeLimit*)
perform a bright star match on the crossmatch results if required by the catalogue search

Key Arguments:

- `matchedObjects` – the list of matched sources from the catalogue crossmatch
- `catalogueName` – the name of the catalogue the crossmatch results from
- `magnitudeLimitFilter` – the name of the column containing the magnitude to filter on
- `lowerMagnitudeLimit` – the lower magnitude limit to match general galaxies against
- `upperMagnitudeLimit` – the upper magnitude limit to match general galaxies against

Return:

- `galaxyMatches` – the trimmed matched sources (associated galaxies only)

Todo:

- update key arguments values and definitions with defaults
- update return values and definitions
- update usage examples and text
- update docstring text
- check sublime snippet exists

- clip any useful text to docs mindmap
- regenerate the docs and check redendering of this docstring

---

physical_separation_crossmatch_against_catalogue(*objectList*, *searchPara*, *search_name*, *brightnessFilter=False*, *classificationType=False*)

perform an physical separation crossmatch against a given catalogue in the database

This search is basically the same as the angular separation search except extra filtering is done to exclude sources outside the physical search radius (matched sources require distance info to calulate physical separations)

Key Arguments:

- `objectList` – transients to be crossmatched
- `searchPara` – parameters of the search (from settings file)
- `search_name` – the name of the search
- `brightnessFilter` – is this search to be constrained by magnitude of the catalogue sources? Default *False*. [bright|faint|general]
- `classificationType` – synonym, association or annotation. Default *False*

Return:

- matchedObjects – any sources matched against the object

To run a physical separation crossmatch, run in a similar way to the angular separation crossmatch:

Usage:

```
search_name = "ned spec sn"
searchPara = self.settings["search algorithm"][search_name]
matchedObjects = xmatcher.physical_separation_crossmatch_against_
↪catalogue(
    objectList=transients,
    searchPara=searchPara,
    search_name=search_name
)
```

Todo:

- update key arguments values and definitions with defaults
- update return values and definitions
- update usage examples and text
- update docstring text
- check sublime snippet exists
- clip any useful text to docs mindmap
- regenerate the docs and check redendering of this docstring

---

8.3.3.2.12 sherlock.transient_classifier (*class*)

class sherlock.**transient_classifier**(*log*, *settings=False*, *update=False*, *ra=False*, *dec=False*, *name=False*, *verbose=0*, *updateNed=True*, *daemonMode=False*, *updatePeakMags=True*, *oneRun=False*)

The Sherlock Transient Classifier

Key Arguments:

- `log` – logger
- `settings` – the settings dictionary
- `update` – update the transient database with crossmatch results (boolean)
- `ra` – right ascension of a single transient source. Default *False*
- `dec` – declination of a single transient source. Default *False*
- `name` – the ID of a single transient source. Default *False*
- `verbose` – amount of details to print about crossmatches to stdout. 0|1|2 Default *0*
- `updateNed` – update the local NED database before running the classifier. Classification will not be as accuracte the NED database is not up-to-date. Default *True*.
- `daemonMode` – run sherlock in daemon mode. In daemon mode sherlock remains live and classifies sources as they come into the database. Default *True*
- `updatePeakMags` – update peak magnitudes in human-readable annotation of objects (can take some time - best to run occationally)
- `oneRun` – only process one batch of transients, usful for unit testing. Default *False*

Usage:

To setup your logger, settings and database connections, please use the `fundamentals` package (see tutorial here).

To initiate a transient_classifier object, use the following:

Todo:

- update the package tutorial if needed

The sherlock classifier can be run in one of two ways. The first is to pass into the coordinates of an object you wish to classify:

```
from sherlock import transient_classifier
classifier = transient_classifier(
    log=log,
    settings=settings,
    ra="08:57:57.19",
    dec="+43:25:44.1",
    name="PS17gx",
    verbose=0
)
classifications, crossmatches = classifier.classify()
```

The crossmatches returned are a list of dictionaries giving details of the crossmatched sources. The classifications returned are a list of classifications resulting from these crossmatches. The lists are ordered from most to least likely classification and the indicies for the crossmatch and the classification lists are synced.

The second way to run the classifier is to not pass in a coordinate set and therefore cause sherlock to run the classifier on the transient database referenced in the sherlock settings file:

```
from sherlock import transient_classifier
classifier = transient_classifier(
    log=log,
    settings=settings,
    update=True
)
classifier.classify()
```

Here the transient list is selected out of the database using the `transient query` value in the settings file:

```
database settings:
    transients:
        user: myusername
        password: mypassword
        db: nice_transients
        host: 127.0.0.1
        transient table: transientBucket
        transient query: "select primaryKeyId as 'id', transientBucketId␣
↪as 'alt_id', raDeg 'ra', decDeg 'dec', name 'name',␣
↪sherlockClassification as 'object_classification'
            from transientBucket where object_classification is null
        transient primary id column: primaryKeyId
        transient classification column: sherlockClassification
        tunnel: False
```

By setting `update=True` the classifier will update the `sherlockClassification` column of the `transient table` with new classification and populate the `sherlock_crossmatches` table with key details of the crossmatched sources from the catalogues database. By setting `update=False` results are printed to stdout but the database is not updated (useful for dry runs and testing new algorithms),

---

Todo:

- update key arguments values and definitions with defaults
- update return values and definitions
- update usage examples and text
- update docstring text
- check sublime snippet exists
- clip any useful text to docs mindmap
- regenerate the docs and check redendering of this docstring

---

Methods

classification_annotations()	*add a detialed classification annotation to each classification in the sherlock_classifications table*
classify()	*classify the transients selected from the transient selection query in the settings file or passed in via the CL or other code*
update_classification_annotations_and_summaries()	*update classification annotations and summaries*
update_peak_magnitudes()	*update peak magnitudes*
_consolidate_coordinateList(coordinateList)	*match the coordinate list against itself with the parameters of the NED search queries to minimise duplicated NED queries*
_create_tables_if_not_exist()	*create the sherlock helper tables if they don't yet exist*
_crossmatch_transients_against_catalogues()	run the transients through the crossmatch algorithm in the settings file
_get_transient_metadata_from_database_list()	use the transient query in the settings file to generate a list of transients to corssmatch and classify
_print_results_to_stdout(classifications, ...)	*print the classification and crossmatch results for a single transient object to stdout*
_rank_classifications(crossmatchArray, colMaps)	*rank the classifications returned from the catalogue crossmatcher, annotate the results with a classification rank-number (most likely = 1) and a rank-score (weight of classification)*
_remove_previous_ned_queries(coordinateList)	iterate through the transient locations to see if we have recent local NED coverage of that area already
_update_ned_stream(transientsMetadataList)	update the NED stream within the catalogues database at the locations of the transients
_update_transient_database(crossmatches, ...)	update transient database with classifications and crossmatch results

classify()

classify the transients selected from the transient selection query in the settings file or passed in via the CL or other code

Return:

- crossmatches – list of dictionaries of crossmatched associated sources
- classifications – the classifications assigned to the transients post-crossmatches (dictionary of rank ordered list of classifications)

See class docstring for usage.

Todo:

- update key arguments values and definitions with defaults
- update return values and definitions
- update usage examples and text
- update docstring text
- check sublime snippet exists
- clip any useful text to docs mindmap

- regenerate the docs and check redendering of this docstring

---

_get_transient_metadata_from_database_list()
use the transient query in the settings file to generate a list of transients to corssmatch and classify

Return:

- `transientsMetadataList`

Todo:

- update key arguments values and definitions with defaults
- update return values and definitions
- update usage examples and text
- update docstring text
- check sublime snippet exists
- clip any useful text to docs mindmap
- regenerate the docs and check redendering of this docstring

---

_update_ned_stream(*transientsMetadataList*)
update the NED stream within the catalogues database at the locations of the transients

Key Arguments:

- `transientsMetadataList` – the list of transient metadata lifted from the database.

Todo:

- update key arguments values and definitions with defaults
- update return values and definitions
- update usage examples and text
- update docstring text
- check sublime snippet exists
- clip any useful text to docs mindmap
- regenerate the docs and check redendering of this docstring

---

_remove_previous_ned_queries(*coordinateList*)
iterate through the transient locations to see if we have recent local NED coverage of that area already

Key Arguments:

- `coordinateList` – set of coordinate to check for previous queries

Return:

- `updatedCoordinateList` – coordinate list with previous queries removed

Todo:

- update key arguments values and definitions with defaults

- update return values and definitions
- update usage examples and text
- update docstring text
- check sublime snippet exists
- clip any useful text to docs mindmap
- regenerate the docs and check redendering of this docstring

---

_crossmatch_transients_against_catalogues(*transientsMetadataListIndex*, *colMaps*)
run the transients through the crossmatch algorithm in the settings file

Key Arguments:

- `transientsMetadataListIndex` – the list of transient metadata lifted from the database.
- `colMaps` – dictionary of dictionaries with the name of the database-view (e.g. `tcs_view_agn_milliquas_v4_5`) as the key and the column-name dictary map as value (`{view_name: {columnMap}}`).

Return:

- `crossmatches` – a list of dictionaries of the associated sources crossmatched from the catalogues database

---

Todo:

- update key arguments values and definitions with defaults
- update return values and definitions
- update usage examples and text
- update docstring text
- check sublime snippet exists
- clip any useful text to docs mindmap
- regenerate the docs and check redendering of this docstring

---

_update_transient_database(*crossmatches*, *classifications*, *transientsMetadataList*, *colMaps*)
update transient database with classifications and crossmatch results

Key Arguments:

- `crossmatches` – the crossmatches and associations resulting from the catlaogue crossmatches
- `classifications` – the classifications assigned to the transients post-crossmatches (dictionary of rank ordered list of classifications)
- `transientsMetadataList` – the list of transient metadata lifted from the database.
- `colMaps` – maps of the important column names for each table/view in the crossmatch-catalogues database

---

Todo:

- update key arguments values and definitions with defaults
- update return values and definitions
- update usage examples and text
- update docstring text
- check sublime snippet exists
- clip any useful text to docs mindmap
- regenerate the docs and check redendering of this docstring

---

_rank_classifications(*crossmatchArray*, *colMaps*)
rank the classifications returned from the catalogue crossmatcher, annotate the results with a classification rank-number (most likely = 1) and a rank-score (weight of classification)

Key Arguments:

- `crossmatchArrayIndex` – the index of list of unranked crossmatch classifications
- `colMaps` – dictionary of dictionaries with the name of the database-view (e.g. `tcs_view_agn_milliquas_v4_5`) as the key and the column-name dictary map as value (`{view_name: {columnMap}}`).

Return:

- `classifications` – the classifications assigned to the transients post-crossmatches
- `crossmatches` – the crossmatches annotated with rankings and rank-scores

---

Todo:

- update key arguments values and definitions with defaults
- update return values and definitions
- update usage examples and text
- update docstring text
- check sublime snippet exists
- clip any useful text to docs mindmap
- regenerate the docs and check redendering of this docstring

---

_print_results_to_stdout(*classifications*, *crossmatches*)
print the classification and crossmatch results for a single transient object to stdout

Key Arguments:

- `crossmatches` – the unranked crossmatch classifications
- `classifications` – the classifications assigned to the transients post-crossmatches (dictionary of rank ordered list of classifications)

Todo:

- update key arguments values and definitions with defaults
- update return values and definitions

- update usage examples and text
- update docstring text
- check sublime snippet exists
- clip any useful text to docs mindmap
- regenerate the docs and check redendering of this docstring

---

_consolidate_coordinateList(*coordinateList*)
match the coordinate list against itself with the parameters of the NED search queries to minimise duplicated NED queries

Key Arguments:

- `coordinateList` – the original coordinateList.

Return:

- `updatedCoordinateList` – the coordinate list with duplicated search areas removed

Usage:

Todo:

- add usage info
- create a sublime snippet for usage
- update package tutorial if needed

```
usage code
```

Todo:

- update key arguments values and definitions with defaults
- update return values and definitions
- update usage examples and text
- update docstring text
- check sublime snippet exists
- clip any useful text to docs mindmap
- regenerate the docs and check redendering of this docstring

---

classification_annotations()
add a detialed classification annotation to each classification in the sherlock_classifications table

Key Arguments: # -

Return:

- None

Usage:

Todo:

- add usage info
- create a sublime snippet for usage
- write a command-line tool for this method
- update package tutorial with command-line tool info if needed

```
usage code
```

Todo:

- update key arguments values and definitions with defaults
- update return values and definitions
- update usage examples and text
- update docstring text
- check sublime snippet exists
- clip any useful text to docs mindmap
- regenerate the docs and check redendering of this docstring

update_classification_annotations_and_summaries(*updatePeakMagnitudes=True*, *cl=False*, *crossmatches=False*)

update classification annotations and summaries

Key Arguments:

- `updatePeakMagnitudes` – update the peak magnitudes in the annotations to give absolute magnitudes. Default *True*
- `cl` – reporting only to the command-line, do not update database. Default *False*

Return:

- None

Usage:

Todo:

- add usage info
- create a sublime snippet for usage
- write a command-line tool for this method
- update package tutorial with command-line tool info if needed

```
usage code
```

Todo:

- update key arguments values and definitions with defaults
- update return values and definitions

- update usage examples and text
- update docstring text
- check sublime snippet exists
- clip any useful text to docs mindmap
- regenerate the docs and check redendering of this docstring

---

update_peak_magnitudes()
update peak magnitudes

Key Arguments: # -

Return:

- None

Usage:

Todo:

- add usage info
- create a sublime snippet for usage
- write a command-line tool for this method
- update package tutorial with command-line tool info if needed

---

```
usage code
```

Todo:

- update key arguments values and definitions with defaults
- update return values and definitions
- update usage examples and text
- update docstring text
- check sublime snippet exists
- clip any useful text to docs mindmap
- regenerate the docs and check redendering of this docstring

---

_create_tables_if_not_exist()
create the sherlock helper tables if they don't yet exist

Key Arguments: # -

Return:

- None

Usage:

Todo:

- add usage info
- create a sublime snippet for usage
- write a command-line tool for this method
- update package tutorial with command-line tool info if needed

```
usage code
```

Todo:

- update key arguments values and definitions with defaults
- update return values and definitions
- update usage examples and text
- update docstring text
- check sublime snippet exists
- clip any useful text to docs mindmap
- regenerate the docs and check redendering of this docstring

generate_match_annotation(*match*, *updatePeakMagnitudes=False*)

generate a human readale annotation for the transient-catalogue source match

Key Arguments:

- `match` – the source crossmatched against the transient
- `updatePeakMagnitudes` – update the peak magnitudes in the annotations to give absolute magnitudes. Default *False*

Return:

- None

Usage:

```
`python usage code `
```

—

```
“‘eval_rst .. todo:

    - add usage info
    - create a sublime snippet for usage
    - write a command-line tool for this method
    - update package tutorial with command-line tool info if needed

“‘
```

8.3.3.3 Functions

sherlock.commonutils.get_crossmatch_catalogues_column_map	*Query the sherlock-catalogues helper tables to generate a map of the important columns of each catalogue*

8.3.3.3.1 sherlock.commonutils.get_crossmatch_catalogues_column_map (*function*)

sherlock.commonutils.**get_crossmatch_catalogues_column_map**(*dbConn*, *log*)

Query the sherlock-catalogues helper tables to generate a map of the important columns of each catalogue

Within your sherlock-catalogues database you need to manually map the inhomogeneous column-names from the sherlock-catalogues to an internal homogeneous name-set which includes *ra*, *dec*, *redshift*, *object name*, *magnitude*, *filter* etc. The column-name map is set within the two database helper tables called `tcs_helper_catalogue_views_info` and `tcs_helper_catalogue_views_info`. See the *'Checklist for Adding A New Reference Catalogue to the Sherlock Catalogues Database'* for more information.

Todo:

- write a checklist for adding a new catalogue to the sherlock database and reference it from here (use the image below of the tcs_helper_catalogue_views_info table)


tcs_helper_catalogue_tables_info
id SMALLINT(5)
table_name VARCHAR(100)
description VARCHAR(60)
url VARCHAR(255)
number_of_rows BIGINT(20)
reference_url VARCHAR(200)
reference_text VARCHAR(70)
notes TEXT
vizier_link VARCHAR(200)
in_ned TINYINT(4)
object_types VARCHAR(100)
version_number VARCHAR(45)
last_updated DATETIME
legacy_table TINYINT(4)
old_table_name VARCHAR(100)
raColName VARCHAR(45)
decColName VARCHAR(45)
catalogue_object_subtypeColName VARCHAR(45)
catalogue_object_idColName VARCHAR(100)
zColName VARCHAR(100)
distanceColName VARCHAR(45)
object_type_accuracy TINYINT(2)
semiMajorColName VARCHAR(100)
semiMajorToArcsec FLOAT
transientStream TINYINT(4)
photoZColName VARCHAR(45)
photoZErrColName VARCHAR(45)
UColName VARCHAR(45)
UErrColName VARCHAR(45)
BColName VARCHAR(45)
BErrColName VARCHAR(45)
VColName VARCHAR(45)
VErrColName VARCHAR(45)
RColName VARCHAR(45)
RErrColName VARCHAR(45)
IColName VARCHAR(45)
IErrColName VARCHAR(45)
JColName VARCHAR(45)
JErrColName VARCHAR(45)
HColName VARCHAR(45)
HErrColName VARCHAR(45)
KColName VARCHAR(45)
KErrColName VARCHAR(45)
_uColName VARCHAR(45)
_uErrColName VARCHAR(45)
_gColName VARCHAR(45)
_gErrColName VARCHAR(45)
_rColName VARCHAR(45)
_rErrColName VARCHAR(45)
_iColName VARCHAR(45)
_iErrColName VARCHAR(45)
_zColName VARCHAR(45)
_zErrColName VARCHAR(45)
_yColName VARCHAR(45)
_yErrColName VARCHAR(45)
unkMagColName VARCHAR(45)
unkMagErrColName VARCHAR(45)
GColName VARCHAR(45)
GErrColName VARCHAR(45)
Indexes
PRIMARY

Key Arguments:

- `dbConn` – the sherlock-catalogues database connection
- `log` – logger

Return:

- `colMaps` – dictionary of dictionaries with the name of the database-view (e.g. `tcs_view_agn_milliquas_v4_5`) as the key and the column-name dictary map as value (`{view_name: {columnMap}}`).

Usage:

To collect the column map dictionary of dictionaries from the catalogues database, use the `get_crossmatch_catalogues_column_map` function:

```
from sherlock.commonutils import get_crossmatch_catalogues_column_map
colMaps = get_crossmatch_catalogues_column_map(
    log=log,
    dbConn=cataloguesDbConn
)
```

8.4 Indexes

- Module Index
- Full Index

8.5 Todo

- Todolist

Index

Symbols

_add_htmids_to_database_table() (*sherlock.imports._base_importer method*), 45
_add_htmids_to_database_table() (*sherlock.imports.ifs method*), 46
_add_htmids_to_database_table() (*sherlock.imports.marshall method*), 48
_add_htmids_to_database_table() (*sherlock.imports.ned method*), 52
_add_htmids_to_database_table() (*sherlock.imports.ned_d method*), 56
_add_htmids_to_database_table() (*sherlock.imports.veron method*), 59
_annotate_crossmatch_with_value_added_parameters() (*sherlock.transient_catalogue_crossmatch method*), 63
_base_importer (*class in sherlock.imports*), 44
_bright_star_match() (*sherlock.transient_catalogue_crossmatch method*), 64
_checkServer() (*sherlock.database method*), 41
_clean_up_columns() (*sherlock.database_cleaner method*), 43
_clean_up_columns() (*sherlock.imports.ned_d method*), 54
_consolidate_coordinateList() (*sherlock.transient_classifier method*), 72
_count_galaxies_requiring_metadata() (*sherlock.imports.ned_d method*), 55
_count_ned_sources_in_database_requiring_metadata() (*sherlock.imports.ned method*), 52
_create_dictionary_of_IFS() (*sherlock.imports.ifs method*), 46
_create_dictionary_of_marshall() (*sherlock.imports.marshall method*), 48
_create_dictionary_of_ned() (*sherlock.imports.ned method*), 51
_create_dictionary_of_ned_d() (*sherlock.imports.ned_d method*), 54
_create_dictionary_of_veron() (*sherlock.imports.veron method*), 59
_create_md_tables() (*sherlock.commonutils.update_wiki_pages method*), 38
_create_tables_if_not_exist() (*sherlock.transient_classifier method*), 74
_crossmatch_transients_against_catalogues() (*sherlock.transient_classifier method*), 70
_do_ned_namesearch_queries_and_add_resulting_metada (*sherlock.imports.ned method*), 51
_download_ned_source_metadata() (*sherlock.imports.ned method*), 51
_galaxy_association_cuts() (*sherlock.transient_catalogue_crossmatch method*), 64
_get_3000_galaxies_needing_metadata() (*sherlock.imports.ned_d method*), 55
_get_metadata_for_galaxies() (*sherlock.imports.ned_d method*), 55
_get_ned_sources_needing_metadata() (*sherlock.imports.ned method*), 51
_get_stream_view_infos() (*sherlock.commonutils.update_wiki_pages method*), 38
_get_table_infos() (*sherlock.commonutils.update_wiki_pages method*), 38
_get_transient_metadata_from_database_list() (*sherlock.transient_classifier method*), 69
_get_view_infos() (*sherlock.commonutils.update_wiki_pages method*), 38
_print_results_to_stdout() (*sherlock.transient_classifier method*), 71
_query_ned_and_add_results_to_database() (*sherlock.imports.ned_d method*), 56
_rank_classifications() (*sherlock.transient_classifier method*), 71
_remove_previous_ned_queries() (*sher-*

lock.transient_classifier method), 69
_setup_tunnel() (*sherlock.database method*), 40
_update_database_helper_table() (*sherlock.imports._base_importer method*), 45
_update_database_helper_table() (*sherlock.imports.ifs method*), 46
_update_database_helper_table() (*sherlock.imports.marshall method*), 48
_update_database_helper_table() (*sherlock.imports.ned method*), 52
_update_database_helper_table() (*sherlock.imports.ned_d method*), 56
_update_database_helper_table() (*sherlock.imports.veron method*), 59
_update_github() (*sherlock.commonutils.update_wiki_pages method*), 38
_update_ned_query_history() (*sherlock.imports.ned method*), 51
_update_ned_stream() (*sherlock.transient_classifier method*), 69
_update_sdss_coverage() (*sherlock.imports.ned_d method*), 56
_update_tcs_helper_catalogue_tables_info_with_new_tables() (*sherlock.database_cleaner method*), 42
_update_tcs_helper_catalogue_views_info_with_new_views() (*sherlock.database_cleaner method*), 43
_update_transient_database() (*sherlock.transient_classifier method*), 70
_updated_row_counts_in_tcs_helper_catalogue_tables_info() (*sherlock.database_cleaner method*), 42
_write_wiki_pages() (*sherlock.commonutils.update_wiki_pages method*), 38

A

add_data_to_database_table() (*sherlock.imports._base_importer method*), 44
add_data_to_database_table() (*sherlock.imports.ifs method*), 46
add_data_to_database_table() (*sherlock.imports.marshall method*), 48
add_data_to_database_table() (*sherlock.imports.ned method*), 52
add_data_to_database_table() (*sherlock.imports.ned_d method*), 57
add_data_to_database_table() (*sherlock.imports.veron method*), 59
angular_crossmatch_against_catalogue() (*sherlock.transient_catalogue_crossmatch method*), 61

C

catalogue_conesearch (*class in sherlock*), 34
classification_annotations() (*sherlock.transient_classifier method*), 72
classify() (*sherlock.transient_classifier method*), 68
clean() (*sherlock.database_cleaner method*), 42
connect() (*sherlock.database method*), 40

D

database (*class in sherlock*), 39
database_cleaner (*class in sherlock*), 41

G

generate_match_annotation() (*sherlock.transient_classifier method*), 75
get_crossmatch_catalogues_column_map() (*in module sherlock.commonutils*), 76

I

ifs (*class in sherlock.imports*), 45
ingest() (*sherlock.imports.ifs method*), 46
ingest() (*sherlock.imports.marshall method*), 48
ingest() (*sherlock.imports.ned method*), 50
ingest() (*sherlock.imports.ned_d method*), 54
ingest() (*sherlock.imports.veron method*), 59

M

marshall (*class in sherlock.imports*), 47
match() (*sherlock.transient_catalogue_crossmatch method*), 61

N

ned (*class in sherlock.imports*), 49
ned_d (*class in sherlock.imports*), 53

P

physical_separation_crossmatch_against_catalogue() (*sherlock.transient_catalogue_crossmatch method*), 65

S

search() (*sherlock.catalogue_conesearch method*), 37
sherlock.commonutils (*module*), 33
sherlock.imports (*module*), 33

T

transient_catalogue_crossmatch (*class in sherlock*), 60
transient_classifier (*class in sherlock*), 66

U

update() (*sherlock.commonutils.update_wiki_pages method*), 38
update_classification_annotations_and_summaries() (*sherlock.transient_classifier method*), 73

update_peak_magnitudes() (*sherlock.transient_classifier method*), 74
update_wiki_pages (*class in sherlock.commonutils*), 37

V

veron (*class in sherlock.imports*), 57