8 Sherlock Tutorial

8.1 Initialisation and Setup

8.1.1 Populating Sherlock’s Settings File

8.1.1.1 Database Settings

8.1.1.2 The Search Algorithm

8.2 Classifying Transients

8.2.1 A Single Transient Classification

8.2.2 Classifying Transients in a Transient Database

8.3 Sherlock’s Catalogue Database

8.3.1 Database Table Naming Scheme

8.3.2 Maintinance and Updates of Catalogues Database

8.3.3 Updating Catalogues and Adding New Catalogues to the Database

8.3.3.1 Subpackages

8.3.3.2 Classes

8.3.3.3 Functions
8.3.3.1 sherlock.commonutil.get_crossmatch_catalogues_column_map (function) 75
8.4 Indexes ................................................................. 77
8.5 Todo ...................................................................... 77
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**

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

**Functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sherlock.commonutils.get_crossmatch_catalogues_column_map</code></td>
<td>Query the sherlock-catalogues helper tables to generate a map of the important columns of each catalogue</td>
</tr>
</tbody>
</table>
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 --name 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 wherever 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

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

.. todo ::

   - document cl_utils module
   - tidy usage text

Usage:

sherlock init
sherlock info [-s <pathToSettingsFile>]
sherlock [-NA] dbmatch [--update] [-s <pathToSettingsFile>]
sherlock [-vN] match -- <ra> <dec> [...pathToSettingsFile]
sherlock clean [-s <pathToSettingsFile>]
sherlock wiki [-s <pathToSettingsFile>]
sherlock import ned <ra> <dec> <radiusArcsec> [...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
...
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stream</td>
<td>Download/stream new data from a given source catalogue into the sherlock sherlock-catalogues database</td>
</tr>
<tr>
<td>info</td>
<td>Print an overview of the current catalogues, views and streams in the sherlock database ready for crossmatching</td>
</tr>
<tr>
<td>ra</td>
<td>The right-ascension coordinate with which to perform a conesearch (sexagesimal or decimal degrees)</td>
</tr>
<tr>
<td>dec</td>
<td>The declination coordinate with which to perform a conesearch (sexagesimal or decimal degrees)</td>
</tr>
<tr>
<td>radiusArcsec</td>
<td>Radius in arcsec of the footprint to download from the online NED database</td>
</tr>
<tr>
<td>cat_name</td>
<td>Name of the catalogue being imported (veron</td>
</tr>
<tr>
<td>stream_name</td>
<td>Name of the stream to import into the sherlock-catalogues database</td>
</tr>
<tr>
<td>database</td>
<td>Database (ifs)</td>
</tr>
</tbody>
</table>

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

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

(continues on next page)
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:
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:

```markdown
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.

```markdown
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.png
```

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:
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 Algorithm

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).

```yaml
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:

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

(continues on next page)
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 continues 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 our 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>
<thead>
<tr>
<th>Table Type</th>
<th>Prefix</th>
<th>Notes</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Catalogue</td>
<td>tcs_cat</td>
<td>The table is named with the scheme tcs_cat_&lt;catalogue name&gt; &lt;version&gt;</td>
<td>tcs_cat_ned_d_v10_2_0</td>
</tr>
<tr>
<td>View</td>
<td>tcs_view</td>
<td>The view is named with the scheme tcs_view_&lt;object type contained&gt; &lt;source table name&gt;</td>
<td>tcs_view_galaxies_ned_d</td>
</tr>
<tr>
<td>Helper</td>
<td>tcs_helper</td>
<td>Mostly used to store relational information, notes on database tables and book-keeper info</td>
<td>tcs_helper_catalogue_tables_info</td>
</tr>
<tr>
<td>Legacy</td>
<td>legacy_tcs_</td>
<td>Legacy tables used in previous incarnations of the transient classifier</td>
<td>legacy_tcs_cat_md01_chiappetti2005</td>
</tr>
</tbody>
</table>

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 its 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:

```
> 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:
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’).
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 --name 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 wherever 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 ::
   - document cl_utils module
   - tidy usage text

Usage:
sherlock init
sherlock info [-s <pathToSettingsFile>]
sherlock [-NA] dbmatch [--update] [-s <pathToSettingsFile>]
sherlock [-vN] 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
```

(continues on next page)
stream: download/stream new data from a given source catalogue
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 cone-search (sexagesimal or decimal degrees)
dec: the declination coordinate with which to perform a cone-search (sexagesimal or decimal degrees)
radiusArcsec: radius in arcsec of the footprint to download from the online NED database
name of the catalogue being imported (veron|ned_d)
name of the stream to import into the sherlock-catalogues database
-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
-v, --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

Chapter 6. Command-Line Usage
Documentation for sherlock is hosted by Read the Docs (last stable version and latest version).
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:

```bash
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

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

(continues on next 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:
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 (\(> 14 − 16^\text{th} \text{ mag}\)) 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?title=Sherlock%20synonyms%20and%20associations%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:
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 Algorithm

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).

```yaml
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:

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

(continues on next page)
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 database to help deal with catalogue versioning (as updated versions of 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 and the views](Crossmatch Catalogue Tables) found on those tables.

8.3.1.1 Table Classes

There are 4 classes of tables in the crossmatch_catalogues database:

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

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:

```
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:
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 ENDING 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

8.3.3.1.2 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

8.3.3.1.3 sherlock.imports (subpackage)

8.3.3.1.4 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 locations 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 catalogues into sherlock-catalogues database
### 8.3.3.2 Classes

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

#### 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 (sexagesimal or decimal degrees, J2000, single location or list of locations)
- `dec` – dec of transient location (sexagesimal 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 required with the cone-search. Default False
• lowerMagnitudeLimit – the lower magnitude limit if a magnitude cut is required with the cone-search. Default False
• magnitudeLimitFilter – the filter to use for the magnitude limit if required. 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 manner:

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

```python
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=transientsDbConn,
    nearestOnly=True,
    physicalSearch=False
)
```
# catalogueMatches ARE ORDERED BY ANGULAR SEPARATION

```python
indices, catalogueMatches = cs.search()
```

print catalogueMatches

The output of this search is:

```python
[{'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 sexagesimal or decimal degrees (J200). It can also accept lists of coordinates:

```python
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)

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

The output of this search is:

```python
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
8.3.3.2.2 Methods

**search()**

*trigger the coneselect*

```python
def search()
    # trigger the coneselect

    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

8.3.3.2.3 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:

```python
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

8.3.3.2.4 Methods

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>update()</strong></td>
<td>Update wiki pages</td>
</tr>
<tr>
<td>_create_md_tables(tableData, viewData, ...)</td>
<td>generate markdown format tables from the database query results</td>
</tr>
<tr>
<td>_get_stream_view_infos(trimmed)</td>
<td>query the sherlock-catalogues database streamed data tables’ metadata</td>
</tr>
<tr>
<td>_get_table_infos(trimmed)</td>
<td>query the sherlock-catalogues database table metadata</td>
</tr>
<tr>
<td>_get_view_infos(trimmed)</td>
<td>query the sherlock-catalogues database view metadata</td>
</tr>
<tr>
<td>_update_github()</td>
<td>commit the changes and push them to github</td>
</tr>
<tr>
<td>_write_wiki_pages()</td>
<td>write the markdown format content of the database tables’ metadata to local wiki pages</td>
</tr>
</tbody>
</table>

**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 formatted content of the database tables’ metadata to local wiki pages

**_update_github()**
commit the changes and push them to github
8.3.3.2.5 sherlock.database (class)

```python
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:

```python
# SETUP ALL DATABASE CONNECTIONS
from sherlock import database

log = ... # Replace with logger
settings = ... # Replace with settings

db = database(log=log, settings=settings)

dbConns, dbVersions = db.connect()

transientsDbConn = dbConns['transients']
cataloguesDbConn = dbConns['catalogues']
marsDbConn = 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

8.3.3.2.6 Methods
connect()  
connect to the various databases, the credentials 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

8.3.3.2.7 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()
• update usage examples and text
• update docstring text
• check sublime snippet exists
• clip any useful text to docs mindmap
• regenerate the docs and check redndering of this docstring

8.3.3.2.8 Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clean()</td>
<td>clean up and run some maintance tasks on the cross-match catalogue helper tables</td>
</tr>
<tr>
<td>_clean_up_columns()</td>
<td>clean up columns</td>
</tr>
<tr>
<td>_update_tcs_helper_catalogue_tables_info</td>
<td>update tcs helper catalogue tables info with new tables</td>
</tr>
<tr>
<td>_update_tcs_helper_catalogue_views_info</td>
<td>update tcs helper catalogue views info with new tables</td>
</tr>
<tr>
<td>_updated_row_counts_in_tcs_helper_catalogue_info</td>
<td>updated row counts in tcs catalogue tables</td>
</tr>
</tbody>
</table>

**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 rednering 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 rednering 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
8.3.3.2.9 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 catalogues 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 cone search. Default False

Usage:

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

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

8.3.3.2.10 Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add_data_to_database_table (dictList[, ...])</td>
<td>Import data in the list of dictionaries in the requested database table</td>
</tr>
<tr>
<td>_add_htmids_to_database_table ()</td>
<td>Add HTMIDs to database table once all the data has been imported (HTM Levels 10,13,16)</td>
</tr>
<tr>
<td>_update_database_helper_table ()</td>
<td>Update the sherlock catalogues database helper table with the time-stamp of when this catalogue was last updated</td>
</tr>
</tbody>
</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 catalogue 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:

```python
self.add_data_to_database_table(
    dictList=dictList,
    createStatement=False
)
```
Todo:

- Write a checklist for creating a new sherlock database importer

```python
_createStatement = _createStatement
```

```python
 Todo:
```

```python
_add_htmids_to_database_table()

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

Usage:

```python
self._add_htmids_to_database_table()
```

```python
_update_database_helper_table()

Update the sherlock catalogues database helper table with the time-stamp of when this catalogue was last updated

Usage:

```python
self._update_database_helper_table()
```

8.3.3.2.11 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:

```python
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
8.3.3.12 Methods

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add_data_to_database_table</td>
<td>Import data in the list of dictionaries in the requested database table</td>
</tr>
<tr>
<td>ingest()</td>
<td>Import the IFS catalogue into the sherlock-catalogues database</td>
</tr>
<tr>
<td>_add_htmids_to_database_table()</td>
<td>Add HTMIDs to database table once all the data has been imported (HTM Levels 10,13,16)</td>
</tr>
<tr>
<td>_create_dictionary_of_IFS()</td>
<td>Generate the list of dictionaries containing all the rows in the IFS stream</td>
</tr>
<tr>
<td>_update_database_helper_table()</td>
<td>Update the sherlock catalogues database helper table with the time-stamp of when this catalogue was last updated</td>
</tr>
</tbody>
</table>

**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:**

```python
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:**

```python
...
```

**_update_database_helper_table()**

Update the sherlock catalogues database helper table with the time-stamp of when this catalogue was last updated

**Usage:**

```python
...
```

**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 catalogue 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:**
```python
def add_data_to_database_table(
    dictList=dictList,
    createStatement=createStatement
)
```

**Todo:**
- Write a checklist for creating a new sherlock database importer

### 8.3.3.2.13 sherlock.imports.marshall (class)

**class** `sherlock.imports.marshall` *(log, settings=False, pathToFile=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:
```python
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
### 8.3.3.2.14 Methods

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>add_data_to_database_table(dictList[, ...])</code></td>
<td>Import data in the list of dictionaries in the requested database table</td>
</tr>
<tr>
<td><code>ingest()</code></td>
<td>Ingest the ePESSTO Marshall transient stream into the catalogues database</td>
</tr>
<tr>
<td><code>_add_htmids_to_database_table()</code></td>
<td>Add HTMIDs to database table once all the data has been imported (HTM Levels 10,13,16)</td>
</tr>
<tr>
<td><code>_create_dictionary_of_marshall(...)</code></td>
<td>Create a list of dictionaries containing all the rows in the marshall stream</td>
</tr>
<tr>
<td><code>_update_database_helper_table()</code></td>
<td>Update the sherlock catalogues database helper table with the time-stamp of when this catalogue was last updated</td>
</tr>
</tbody>
</table>

#### `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:**

```python
self._add_htmids_to_database_table()```

#### `_update_database_helper_table()`

Update the sherlock catalogues database helper table with the time-stamp of when this catalogue was last updated

**Usage:**

```python
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 catalogue 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:**

```python
def add_data_to_database_table(
    dictList,  # a list of dictionaries containing all the rows in the catalogue to be imported
    createStatement=None  # the table's MySQL create statement (used to generate table if it does not yet exist in database). Default None
)
```

**Todo:**

- Write a checklist for creating a new sherlock database importer

## 8.3.3.2.15 sherlock.imports.ned (class)

**class** `sherlock.imports.ned`  
**Bases:** `sherlock.imports._base_importer._base_importer`  
**docstring**

*Using a list of coordinates, query the online NED database and import sources found within a given search radius of each of the locations 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. Queries 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:

```python
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

### 8.3.3.2.16 Methods

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>add_data_to_database_table(dictList[, ...])</code></td>
<td>Import data in the list of dictionaries in the requested database table</td>
</tr>
<tr>
<td><code>ingest()</code></td>
<td>Perform conesearches of the online NED database and import the results into a the sherlock-database</td>
</tr>
<tr>
<td><code>_add_htmids_to_database_table()</code></td>
<td>Add HTMIDs to database table once all the data has been imported (HTM Levels 10.13.16)</td>
</tr>
<tr>
<td><code>_count_ned_sources_in_database_requiring_metadata()</code></td>
<td>Count the sources in the NED table requiring metadata</td>
</tr>
<tr>
<td><code>_create_dictionary_of_ned()</code></td>
<td>Create a list of dictionaries containing all the object ids (NED names) in the ned stream</td>
</tr>
<tr>
<td><code>_do_ned_namesearch_queries_and_add_result_metadata_to_database()</code></td>
<td>Query NED via name search and add result metadata to database</td>
</tr>
<tr>
<td><code>_download_ned_source_metadata()</code></td>
<td>Query NED using the names of the NED sources in our local database to retrieve extra metadata</td>
</tr>
<tr>
<td><code>_get_ned_sources_needing_metadata()</code></td>
<td>Get the names of 50000 or less NED sources that still require metadata in the database</td>
</tr>
<tr>
<td><code>_update_database_helper_table()</code></td>
<td>Update the sherlock catalogues database helper table with the time-stamp of when this catalogue was last updated</td>
</tr>
<tr>
<td><code>_update_ned_query_history()</code></td>
<td>Update the database helper table to give details of the ned cone searches performed</td>
</tr>
</tbody>
</table>

**ingest**

`ingest()` performs 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 \texttt{tcs\_cat\_ned\_stream} table

4. Updates the NED query history table

5. Queries NED via NED IDs for the remaining source metadata to be added to the \texttt{tcs\_cat\_ned\_stream} table

Usage:

Having setup the NED object with a coordinate list and cone-search radius, run the \texttt{ingest()} method

\[
\texttt{stream.\_ingest()}
\]

Todo:

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

\_create\_dictionary\_of\_ned()

\textit{Create a list of dictionaries containing all the object ids (NED names) in the ned stream}

Return:

• \texttt{dictList} - a list of dictionaries containing all the object ids (NED names) in the ned stream

Usage:

\[
\texttt{dictList = stream.\_create\_dictionary\_of\_ned()}
\]

\_update\_ned\_query\_history()

\textit{Update the database helper table to give details of the ned cone searches performed}

Usage:

\[
\texttt{stream.\_update\_ned\_query\_history()}
\]

\_download\_ned\_source\_metadata()

\textit{Query NED using the names of the NED sources in our local database to retrieve extra metadata}

Usage:

\[
\texttt{stream.\_download\_ned\_source\_metadata()}
\]

\_get\_ned\_sources\_needing\_metadata()

\textit{Get the names of 50000 or less NED sources that still require metabase in the database}

Return:

• \texttt{len(self.theseIds)} – the number of NED IDs returned

Usage:

\[
\texttt{numberSources = stream.\_get\_ned\_sources\_needing\_metadata()}
\]

\_do\_ned\_namesearch\_queries\_and\_add\_resulting\_metadata\_to\_database(batchCount)

\textit{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:**

```python
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:**

```python
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:**

```python
self._add_htmids_to_database_table()
```

---

__**_update_database_helper_table()**__

*Update the sherlock catalogues database helper table with the time-stamp of when this catalogue was last updated*

**Usage:**

```python
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 catalogue 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:**

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

---

**Todo:**

- Write a checklist for creating a new sherlock database importer
8.3.3.17 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:

```python
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

8.3.3.2.18 Methods

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

Continued on next page
**Table 13 – continued from previous page**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>_update_database_helper_table()</code></td>
<td>Update the sherlock catalogues database helper table with the time-stamp of when this catalogue was last updated</td>
</tr>
<tr>
<td><code>_update_sdss_coverage()</code></td>
<td>update sdss coverage</td>
</tr>
</tbody>
</table>

**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:**

.

__update_database_helper_table__()
Update the sherlock catalogues database helper table with the time-stamp of when this catalogue was last updated
Usage:

```python
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 catalogue 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:

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

**Todo:**

- Write a checklist for creating a new sherlock database importer

---

**8.3.3.2.19 sherlock.imports.veron (class)**

```python
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:

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

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

8.3.3.2.20 Methods

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

Continued on next 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 catalogue 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:**
```python
self._add_htmids_to_database_table()
```

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

**Usage:**
```python
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 catalogue 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:

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

Todo:

- Write a checklist for creating a new sherlock database importer

---

**8.3.3.21 sherlock.transient_catalogue_crossmatch (class)**

The `sherlock.transient_catalogue_crossmatch` class is used to crossmatch a list of transients against a suite of catalogues according to a 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:

```python
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:

```python
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 rendering of this docstring

8.3.3.22 Methods

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

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() perform a physical separation crossmatch against a given catalogue in the database

_annotate_crossmatch_with_value_added_parameters() annotate each crossmatch with physical parameters such as 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 rendering 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

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

transients = ['ps1_designation': 'PS1-13dcr',
              'name': '3I3Phzx',
              'detection_list_id': 2,
              'local_comments': '',
              'ra': 4.754236999477372,
              'followup_id': 1140386L,
              'dec': 28.276703631398625,
              'id': 1001901011281636100L,
              'object_classification': 0L,
              ],

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

62 Chapter 8. Sherlock Tutorial
Then run the `angular_crossmatch_against_catalogue` method to crossmatch against the catalogues and return results:

```python
# 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

```python
_annotate_crossmatch_with_value_added_parameters(crossmatchDict, cata-
   logueName, 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
- 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
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 calculate 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:

```python
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
8.3.3.23 sherlock.transient_classifier (class)

```python
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 accurate 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 occasionally)
- `oneRun` – only process one batch of transients, useful 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:

```python
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()
```

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:
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 indices 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:

```python
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:

```plaintext
database settings
transients
  user: myusername
  password: mypassword
  db: nice_transients
  host: 127.0.0.1
transient bucket: 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
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 rednering of this docstring
### 8.3.3.2.24 Methods

- **classification_annotations()**
  
  *add a detailed 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 crossmatch 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
**_get_transient_metadata_from_database_list_**

use the transient query in the settings file to generate a list of transients to crossmatch 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 dictionary 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 catalogue 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:
**_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 dictionary 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

```python
_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

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

```python
classification_annotations ()

add a detailed 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

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)*

update classification annotations and summaries

**Key Arguments:**

• **updatePeakMagnitudes** – update the peak magnitudes in the annotations to give absolute magnitudes. Default *True*

**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
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

_todo_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 redennering 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

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

8.3.3.3 Functions

```python
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)
<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>SMALLINT</td>
<td></td>
</tr>
<tr>
<td>table_name</td>
<td>VARCHAR</td>
<td></td>
</tr>
<tr>
<td>description</td>
<td>VARCHAR</td>
<td></td>
</tr>
<tr>
<td>url</td>
<td>VARCHAR</td>
<td></td>
</tr>
<tr>
<td>number_of_rows</td>
<td>BIGINT</td>
<td></td>
</tr>
<tr>
<td>reference_url</td>
<td>VARCHAR</td>
<td></td>
</tr>
<tr>
<td>reference_text</td>
<td>VARCHAR</td>
<td></td>
</tr>
<tr>
<td>notes</td>
<td>TEXT</td>
<td></td>
</tr>
<tr>
<td>vizier_link</td>
<td>VARCHAR</td>
<td></td>
</tr>
<tr>
<td>ln_rad</td>
<td>TINYINT</td>
<td></td>
</tr>
<tr>
<td>object_types</td>
<td>VARCHAR</td>
<td></td>
</tr>
<tr>
<td>version_number</td>
<td>VARCHAR</td>
<td></td>
</tr>
<tr>
<td>last_updated</td>
<td>DATETIME</td>
<td></td>
</tr>
<tr>
<td>legacy_table</td>
<td>TINYINT</td>
<td></td>
</tr>
<tr>
<td>old_table_name</td>
<td>VARCHAR</td>
<td></td>
</tr>
<tr>
<td>raColName</td>
<td>VARCHAR</td>
<td></td>
</tr>
<tr>
<td>decColName</td>
<td>VARCHAR</td>
<td></td>
</tr>
<tr>
<td>catalogue_object_subtypeColName</td>
<td>VARCHAR</td>
<td></td>
</tr>
<tr>
<td>catalogue_object_idColName</td>
<td>VARCHAR</td>
<td></td>
</tr>
<tr>
<td>zCoName</td>
<td>VARCHAR</td>
<td></td>
</tr>
<tr>
<td>distanceCoName</td>
<td>VARCHAR</td>
<td></td>
</tr>
<tr>
<td>object_type_accuracy</td>
<td>TINYINT</td>
<td></td>
</tr>
<tr>
<td>semimajorCoName</td>
<td>VARCHAR</td>
<td></td>
</tr>
<tr>
<td>semiminorToAcsecCoName</td>
<td>FLOAT</td>
<td></td>
</tr>
<tr>
<td>transientStream</td>
<td>TINYINT</td>
<td></td>
</tr>
<tr>
<td>photoZCoName</td>
<td>VARCHAR</td>
<td></td>
</tr>
<tr>
<td>photoZErrCoName</td>
<td>VARCHAR</td>
<td></td>
</tr>
<tr>
<td>UCoName</td>
<td>VARCHAR</td>
<td></td>
</tr>
<tr>
<td>UErrCoName</td>
<td>VARCHAR</td>
<td></td>
</tr>
<tr>
<td>BCoName</td>
<td>VARCHAR</td>
<td></td>
</tr>
<tr>
<td>BErrCoName</td>
<td>VARCHAR</td>
<td></td>
</tr>
<tr>
<td>VCoName</td>
<td>VARCHAR</td>
<td></td>
</tr>
<tr>
<td>VErrCoName</td>
<td>VARCHAR</td>
<td></td>
</tr>
<tr>
<td>RC0Name</td>
<td>VARCHAR</td>
<td></td>
</tr>
<tr>
<td>REC0Name</td>
<td>VARCHAR</td>
<td></td>
</tr>
<tr>
<td>ICoName</td>
<td>VARCHAR</td>
<td></td>
</tr>
<tr>
<td>IErColName</td>
<td>VARCHAR</td>
<td></td>
</tr>
<tr>
<td>JCoName</td>
<td>VARCHAR</td>
<td></td>
</tr>
<tr>
<td>JErrColName</td>
<td>VARCHAR</td>
<td></td>
</tr>
<tr>
<td>HCoName</td>
<td>VARCHAR</td>
<td></td>
</tr>
<tr>
<td>HErrColName</td>
<td>VARCHAR</td>
<td></td>
</tr>
<tr>
<td>KCoName</td>
<td>VARCHAR</td>
<td></td>
</tr>
<tr>
<td>KErrColName</td>
<td>VARCHAR</td>
<td></td>
</tr>
<tr>
<td>xCoName</td>
<td>VARCHAR</td>
<td></td>
</tr>
<tr>
<td>xErrColName</td>
<td>VARCHAR</td>
<td></td>
</tr>
<tr>
<td>yCoName</td>
<td>VARCHAR</td>
<td></td>
</tr>
<tr>
<td>yErrColName</td>
<td>VARCHAR</td>
<td></td>
</tr>
<tr>
<td>zCoName</td>
<td>VARCHAR</td>
<td></td>
</tr>
<tr>
<td>zErrColName</td>
<td>VARCHAR</td>
<td></td>
</tr>
<tr>
<td>unkMagCoName</td>
<td>VARCHAR</td>
<td></td>
</tr>
<tr>
<td>unkMagErrColName</td>
<td>VARCHAR</td>
<td></td>
</tr>
<tr>
<td>gCoName</td>
<td>VARCHAR</td>
<td></td>
</tr>
<tr>
<td>gErrColName</td>
<td>VARCHAR</td>
<td></td>
</tr>
</tbody>
</table>
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 dictionary 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:

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

8.4 Indexes

- Module Index
- Full Index

8.5 Todo

- Todolist
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_column() (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_metadata_to_database() (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() (sherlock.commonutils.update_wiki_pages method), 38
V

veron (class in sherlock.imports), 57