Matt's astro python toolbox Documentation

Release 0.1

Matt Craig

July 01, 2016

Contents

1 Contents **1**
- 1.1 Overview . . . 1
- 1.2 Installation . . . 2
- 1.3 Automated header processing . . . 4
- 1.4 Manual header processing . . . 14
- 1.5 Image Management . . . 16
- 1.6 Header processing . . . 23
- 1.7 Index . . . 37

Python Module Index **39**

CHAPTER 1

Contents

1.1 Overview

This package provides two types of functionality; only the first is likely to be of general interest.

1.1.1 Classes for managing a collection of FITS files

The class *ImageFileCollection* provides a table summarizing the values of FITS keywords in the files in a directory and provides easy ways to iterate over the HDUs, headers or data in those files. As a quick example:

```
>>> from msumastro import ImageFileCollection
>>> ic = ImageFileCollection('.', keywords=['imagetyp', 'filter'])
>>> for hdu in ic.hdus(imagetype='LIGHT', filter='I'):
...     print hdu.header, hdu.data.mean()
```

does what you would expect: it loops over all of the images in the collection whose image type is 'LIGHT' and filter is 'I'. For more details see *Image Management*.

The *TableTree* constructs, from the summary table of an *ImageFileCollection*, a tree organized by values in the FITS headers of the collection. See *Image Management* for more details and examples.

1.1.2 Header processing of images from the Feder Observatory

Semi-automatic image processing

Command line scripts for automated updating of FITS header keywords. The intent with these is that they will rarely need to be used manually once a data preparation pipeline is set up.

The simplest option here is to use *run_standard_header_process* which will chain together all of the steps in data preparation for you. When using *run_standard_header_process* consider using the `--scripts-only` option, which generates *bash* scripts to carry out the data preparation. This gives you a complete record of the commands run in addition to the log files that are always generated.

All of these scripts can also be run from your python code if desired.

Manual header or image manipulation

Command lines scripts that easily automate a small number of tasks that occur frequently enough that it is convenient to have them available at the command line instead of requiring that new code to be written each time they are used.

There are currently two examples of this:

- *quick_add_keys_to_file*, for modifying the values of FITS header keywords with minimal effort.
- *sort_files* for *Sorting files based on image properties*

All of these scripts can also be run from your python code if desired.

1.2 Installation

1.2.1 This software

Users

This software requires a python distribution that includes numpy and other packages that support scientific work with python. The easiest way to get these is to download and install the Anaconda python distribution. Note that the Anaconda distribution includes `astropy`.

Install the way you install most python software:

```
pip install msumastro
```

followed (optionally) by:

```
pip install astropysics
```

only if you need the Feder Observatory stuff. You do *not* need astropysics for the image management features likely to be of broadest interest.

Developers

Install this software by downloading a copy from the github page for the code. On Mac/Linux do this by typing, in a terminal in the directory in which you want to run the code:

```
git clone https://github.com/mwcraig/msumastro.git
```

Navigate to the directory in which you downloaded it and run:

```
python setup.py develop
```

With this setup any changes you make to the source code will be immediately available to you without additional steps.

1.2.2 Dependencies

Python

This software has only been tested in python 2.7.x. It does **not** work in 3.x.

Note: Most of the requirements below will be taken care of automatically if you install *msumastro* with `pip` or `setup.py` as described above. The exceptions are numpy and scipy

---

Python packages

Required

Nothing will work without these:

- numpy (*included with anaconda*): If you need to install it, please see the instructions at the SciPy download site. Some functionality may require SciPy.
- astropy (*included with anaconda*): If you need to install it, do so with:

```
pip install astropy
```

Required for some features

Most of the header patching functionality requires astropysics:

- astropysics: Install with:

```
pip install astropysics
```

Very strongly recommended if you want to test your install

- pytest_capturelog: Install with:

```
pip install pytest-capturelog
```

Required to build documentation

You only need to install the packages below if you want to build the documentation yourself:

- numpydoc: Install using either `pip`, or, if you have the Anaconda python distribution, like this:

```
conda install numpydoc
```

- sphinx_argparse: Install it this way:

```
pip install sphinx-argparse
```

(mostly) Non-python software: astrometry.net

Note: There is one piece of python software you need for astrometry.net and for now you need to install it manually:

```
pip install pyfits
```

If you want to be able to use the script *Adding astrometry: run_astromtery.py* you need a local installation of astrometry.net and sextractor (the latter works better than the source detection built into astrometry.net) The easiest way to do that (on a Mac) is with homebrew. Once you have installed homebrew the rest is easy (unless it fails, of course...):

- `brew tap homebrew/science` (only needs to be done once; connects the set of homebrew science formulae)
- `brew install sextractor` (note this can take a a few minutes)

- `brew install --env=std astrometry.net` [Note the option `--env=std`. It is necessary to ensure homebrew sees your python installation.]

1.3 Automated header processing

1.3.1 Overview

The scripts described here are intended primarily to be run in an automated way whenever new images are uploaded to the physics server. Each can also be run manually, either from the command line or from python.

Both these ways of invoking the script (from the command line or from python) is are wrappers around the python functions that do the real work. References to those functions, which provide more control over what you can do at the expense of taking more effort to understand, are provided below where appropriate.

The purpose of the header processing is to:

- Modify or add keywords to the FITS header to make working with other software easier:
  - Standardize names instead using the MaxIm DL defaults (e.g. `RA` and `Dec` instead of `OBJCTRA` and `OBJCTDEC`)
  - Set `IMAGETYP` to IRAF default, *i.e.* "BIAS", "DARK", "FLAT" or "LIGHT"
  - Add convenient keywords that MaxImDL does not always include (e.g. `AIRMASS`, `HA`, `LST`)
  - Add keywords indicating the overscan region, if any, in the image.
- Add astrometry to the FITS header so that RA/Dec can be extracted/displayed for sources in the image.
- Add object names to the HEADER where possible.
- Identify images that need manual action to add any of the information above.
- Create a table of images in each directory with columns for several FITS header keywords to facilitate indexing of images taken at Feder.

1.3.2 Intended workflow

The three primary scripts are, in the order in which they are intended (though not *required*, necessarily) to run:

- `run_patch` to do a first round of header patching that puts site information, LST, airmass (where appropriate) and RA/Dec information (where appropriate) into the files. See *Header patching: run_patch.py* for details.
- `run_triage` (details at *Find problems and create summary: run_triage.py*) to:
- generate a table summarizing properties of images in a directory. Each image is one row in the table and the columns are keywords from the FITS headers.
- create files with lists of images missing key information.
- Fix any problems identified by `run_triage`. The script `quick_add_keys_to_file` may be useful for this; it is an easy way to add/modify the values of keywords in FITS headers from the command line; see details at *Manual intervention: quick_add_keys_to_file.py*. After fixing these problems you may need to re-run patch, particularly if you have added pointing information or changed the `IMAGETYP` of any of the images.
- `run_astrometry` to use astrometry.net to add WCS (astrometry) information to the file. See *Adding astrometry: run_astromtery.py* for details. **Note that this requires a local installation** of astrometry.net.
- If desired, `run_triage` again to regenerate the table of image information.

1.3.3 The intended workflow will not work when...

The workflow above works great when the images that come off the telescope contain pointing information (i.e. RA/Dec), filter information and the image type in the header matches the actual image type.

Manual intervention will be required in any of these circumstances:

- **There is no pointing information in the files.** In files that are produced at Feder Observatory the pointing information is contained in the FITS keywords `OBJCTRA` and `OBJCTDEC`. If there is no pointing information, astrometry will not be added to the image headers. astrometry.net can actually do blind astrometry, but it is fairly time consuming. Alternatives are suggested below.
  - **How to identify this case**: There are two ways this problem may be noted. If *`run_triage`* has been run (and it *is* run in the standard workflow) then a file called "NEEDS_POINTING_INFO.txt" will be created that lists all of the light files missing pointing information. In addition, one file with suffix *.blind* will be created for each light file which contains no pointing information.
- **Filter information is missing for light or flat images.** All of the data preparation will occur if the `FILTER` keyword is missing from the headers for light or flat images, but the filter needs to be added to make the images *useful*.
  - **How to identify this case**: A file called "NEEDS_FILTER.txt" will be created as part of the standard workflow that lists each file that needs filter information.
- **Incorrect image type set for image.** If the incorrect image type is set it can prevent some data preparation steps to be omitted that should actually occur or cause steps to be attempted that shouldn't be. For example, if an image is really a `LIGHT` image but is labeled in the header as a `FLAT`, then no attempt will be made to calculate an apparent position (Alt/Az) for the frame or to add astrometry. If the mistake is reversed, with a `FLAT` image labeled as `LIGHT` an attempt will be made to add astrometry which will fail.
  - **How to identify this case**: Manual inspection of affected images is the only reliable way to do this. A good place to start looking is at light files for which adding astrometry failed, file names whose name implies a different type than its `IMAGETYP` in the FITS header (e.g. a file with `IMAGETYP = LIGHT` whose name is `flat-001R.fit`)
- **The object being observed is not in the master object list.** The standard workflow has run but object names have not been add to all of the light files. This occurs when the object of the image was not in the list of objects used by `run_patch.py` or the object's RA/Dec was too far from the center of the image to be matched.
  - **How to identify this case**: The script `run_triage.py`, part of the standard workflow, will produce a file called "NEEDS_OBJECT.txt" with a list of light files for which there is no object.

1.3.4 Fixes for cases that require intervention

The discussion below is deliberately broad. For some concrete examples see *Manual header processing*

- **Adding pointing information**: There are a few options here:
  - Use *`quick_add_keys_to_file`* to add the `OBJECT` keyword to the header, then `add_ra_dec_from_object_name()` to add pointing information, then *`run_astrometry`* to add astrometry to the images.
  - Use *`quick_add_keys_to_file`* to add the `OBJECT`, `RA`, `DEC`, `OBJCTRA` and `OBJCTDEC` to the headers, then *`run_astrometry`* to add astrometry to the images. This route is **not recommended** because it is easy to use a format for RA/Dec that isn't understood (or is misinterpreted) by the code that adds astrometry.
  - Do blind astrometry to add pointing information, then use `add_object_info()` to add object names. There are no inherent with with this approach, though it may be simpler to add the astrometry

then re-run the standard processing workflow to add any missing keywords than it is to manually use `add_object_info()`

- **Adding filter information**: The hard part here is not adding the filter keyword, it is figuring out what filter was used when the image was taken. You are on your own in figuring out that piece. Once you know what the filter should be, use *quick_add_keys_to_file* to add the keyword `FILTER` to the relevant files.
- **Adding filter information**: As with adding filter information, the hard part is figuring out what the image type *should* be. In practice most cases of this are light images misidentified as flat and *vice versa* and it ought to be easy to determine which of those an image is at a glance (arguably, if you can't tell at a glance then the image is probably useful as neither a light nor a flat image). Once you know what the image type should be, use *quick_add_keys_to_file* to set the keyword `IMAGETYP` to the appropriate value in the relevant files. Allowed values for `IMAGETYP` are "BIAS", "DARK", "FLAT" or "LIGHT".
- **Adding object information**: Assuming pointing information is already in the header for the images that need object information this is fairly straightforward. One way to do it is to add the object to the master object list and run `add_object_info()` (or even just re-run *run_patch*, which will end up re-doing some of the keyword-patching work). Another way to approach is to use *quick_add_keys_to_file* to set the `OBJECT` keyword directly. **Either way you are encouraged to upadte the master object list.**

1.3.5 Detailed list of keywords changed

Keywords purged before further processing

Some **keywords are purged** from the original headers because I don't trust the values that MaxImDL v5 puts in:

```
OBJECT
JD
JD-HELIO
OBJCTALT
OBJCTAZ
OBJCTHA
AIRMASS
```

Keywords modified for all files

The keywords that are currently added/modified by `patch_headers` for **all files** are:

```
IMAGETYP: Type of image
LATITUDE: [degrees] Observatory latitude
LONGITUD: [degrees east] Observatory longitude
ALTITUDE: [meters] Observatory altitude
LST: Local Sidereal Time at start of observation
JD-OBS: Julian Date at start of observation
MJD-OBS: Modified Julian date at start of observation
BIASSEC: Region of the image useful for subtracting overscan
TRIMSEC: Region to which the image should be trimmed after removing overscan
```

Keywords modified only for light files

The keywords that are currently added/modified by `patch_header` for **light files only** are:

```
RA: Approximate RA at EQUINOX
DEC: Approximate DEC at EQUINOX
OBJECT: Target of the observations
```

```
HA: Hour angle
AIRMASS: Airmass (Sec(Z)) at start of observation
ALT-OBJ: [degrees] Altitude of object, no refraction
AZ-OBJ: [degrees] Azimuth of object, no refraction
```

1.3.6 Reference/API

Command-line scripts

Each of the command-line scripts described below is also callable from python. The details of how you call it from python are described below.

Both these ways of invoking the script (from the command line or from python) is are wrappers around the python functions that do the real work. References to those functions, which tend to provide more control over what you can do at the expense of taking more effort to understand, are provided below where appropriate.

Running the complete standard workflow: `run_standard_header_process.py`

Usage summary

```
usage: run_standard_header_process.py [-h]
                                      (--dest-root DEST_ROOT | --overwrite-source)
                                      [--scripts-only] [-r {a,t,p}]
                                      [--no-blind] [-o OBJECT_LIST]
                                      [--quiet-console] [--silent-console]
                                      [--debug] [--quiet-log]
                                      source_root
```

Positional arguments:

source_root All directories below this one that contain images will be processed

Options:

--dest-root If set, image directories below ''source-root'' will be copied into this directory tree. Only directories that contain image files will be copied; any intermediary directories required to contain directories that contain images will also be created.

--overwrite-source=False This flag must be used to overwrite images in the course directory.

--scripts-only=False This script will write a single shell script with the name provided in this option. No images will be modified or directories created, but the script can be run to do those things.

-r, --run-only Select which scripts you want to run. This can be any combination of [p]atch, [a]strometry and [t]riage.

Possible choices: a, t, p

--no-blind=False Disable astrometry for images without pointing information

-o, --object-list Path to or URL of file containing list (and optionally coordinates of) objects that might be in these files. If not provided it defaults to looking for a file called obsinfo.txt in the directory being processed

--quiet-console=False Log only errors (or worse) to console while running scripts

--silent-console=False Turn off all logging output to console

--debug=False Turn on very detailed logging output

--quiet-log=False Log only warnings (or worse) to FILES AND CONSOLE while running scripts

---

Header patching: `run_patch.py`

For a detailed description of which header keywords are modified see *Keywords purged before further processing*.

> **Warning:** This script OVERWRITES the image files in the directories specified on the command line unless you use the –destination-dir option.

Usage summary

```
usage: run_patch.py [-h] [-v] [-d DESTINATION_DIR] [--debug] [-n]
                    [--quiet-console] [--silent-console] [-o OBJECT_LIST]
                    [--overscan-only]
                    dir [dir ...]
```

Positional arguments:

dir Directory to process

Options:

-v=False, --verbose=False provide more information during processing

-d, --destination-dir Directory in which output from this script will be stored

--debug=False Turn on very detailed logging output

-n=False, --no-log-destination=False Do not write log files to destination directory

--quiet-console=False Log only errors (or worse) to console while running scripts

--silent-console=False Turn off all logging output to console

-o=https://raw.github.com/mwcraig/feder-object-list/master/feder_object_list.csv, --object-list=https://raw.github.co
Path to or URL of file containing list (and optionally coordinates of) objects that might be in these files. If not provided it defaults to looking for a file called obsinfo.txt in the directory being processed

--overscan-only=False Only add appropriate overscan keywords

DESCRIPTION For each directory provided on the command line the headers all of the FITS files in that directory are modified to add information like LST, apparent object position, and more. See the full documentation for a list of the specific keywords that are modified.

Header patching This is basically a wrapper around the function `patch_headers.patch_headers()` with the options set so that:

- "Bad" keywords written by MaxImDL 5 are purged.
- `IMAGETYP` keyword is changed from default MaxIM DL style to IRAF style (e.g. "Bias Frame" to "BIAS")

- Additional useful times like LST, JD are added to the header.
- Apparent position (Alt/Az, hour angle) are added to the header.
- Information about overscan is added to the header.
- Files are overwritten.

For more control over what is patched and where the patched files are saved see the documentation for `patch_headers` at `patch_headers.patch_headers()`.

Adding OBJECT keyword `run_patch` also adds the name of the object being observed when appropriate (i.e. only for light files) and possible. It needs to be given a list of objects; looking up the coordinates for those objects requires an Internet connection. See

For a detailed description of the object list file see `Object file format`.

for a detailed description of the function that actually adds the object name see `patch_headers.add_object_info()`.

If no object list is specified or present in the directory being processed the *OBJECT* keyword is simply not added to the FITS header.

Note: This script is **NOT RECURSIVE**; it will not process files in subdirectories of the the directories supplied on the command line.

Warning: This script OVERWRITES the image files in the directories specified on the command line unless you use the –destination-dir option.

EXAMPLES Invoking this script from the command line:

```
run_patch.py /my/folder/of/images
```

To work on the same folder from within python, do this:

```
from msumastro.scripts import run_patch
run_patch.main(['/my/folder/of/images'])
```

To use the same object list for several different directories do this:

```
run_patch.py --object-list path/to/list.txt dir1 dir2 dir3
```

where `path/to/list.txt` is the path to your object list and `dir1`, `dir2`, etc. are the directories you want to process.

From within python this would be:

```
from msumastro.scripts import run_patch
run_patch.main(['--object-list', 'path/to/list.txt',
                'dir1', 'dir2', 'dir3'])
```

Adding astrometry: `run_astromtery.py`

Warning: This script OVERWRITES the image files in the directories specified on the command line unless you use the –destination-dir option.

Usage summary

```
usage: run_astrometry.py [-h] [-v] [-d DESTINATION_DIR] [--debug] [-n]
                         [--quiet-console] [--silent-console] [-b]
                         dir [dir ...]
```

Positional arguments:

dir Directory to process

Options:

-v=False, --verbose=False provide more information during processing

-d, --destination-dir Directory in which output from this script will be stored

--debug=False Turn on very detailed logging output

-n=False, --no-log-destination=False Do not write log files to destination directory

--quiet-console=False Log only errors (or worse) to console while running scripts

--silent-console=False Turn off all logging output to console

-b=False, --blind=False Turn ON blind astrometry; disabled by default because it is so slow.

DESCRIPTION

For each directory provided on the command line add astrometry to the light files (those with `IMAGETYP='LIGHT'` in the FITS header).

By default, astrometry is added only for those files with pointing information in the header (specifically, RA and Dec) because blind astrometry is fairly slow. It may be faster to insert RA/Dec into those files before doing astrometry.

The functions called by this script set the WCS reference pixel to the center of the image, which turns out to make aligning images a little easier.

For more control over the parameters see `add_astrometry()` and for even more control, `call_astrometry()`.

Note: This script is **NOT RECURSIVE**; it will not process files in subdirectories of the the directories supplied on the command line.

Warning: This script OVERWRITES the image files in the directories specified on the command line unless you use the –destination-dir option.

EXAMPLES

Invoking this script from the command line:

```
run_astrometry.py /my/folder/of/images
```

To work on the same folder from within python, do this:

```
from msumastro.scripts import run_astrometry
run_astrometry.main(['/my/folder/of/images'])
```

Find problems and create summary: `run_triage.py`

Usage summary

```
usage: run_triage.py [-h] [--debug] [-v] [-d DESTINATION_DIR] [-n]
                     [--quiet-console] [--silent-console] [-k KEY] [-l] [-a]
                     [-t TABLE_NAME] [-o OBJECT_NEEDED_LIST]
                     [-p POINTING_NEEDED_LIST] [-f FILTER_NEEDED_LIST]
                     [-y ASTROMETRY_NEEDED_LIST]
                     [dir [dir ...]]
```

Positional arguments:

dir Directory to process

Options:

--debug=False Turn on very detailed logging output

-v=False, --verbose=False provide more information during processing

-d, --destination-dir Directory in which output from this script will be stored

-n=False, --no-log-destination=False Do not write log files to destination directory

--quiet-console=False Log only errors (or worse) to console while running scripts

--silent-console=False Turn off all logging output to console

-k=[], --key=[] FITS keyword to add to table in addition to the defaults; for multiple keywords use this option multiple times.

-l=False, --list-default=False Print default list keywords put into table and exit

-a=False, --all=False Construct table from all FITS keywords present in headers and the list of default keywords.

-t=Manifest.txt, --table-name=Manifest.txt Name of file in which table is saved; default is Manifest.txt

-o=NEEDS_OBJECT_NAME.txt, --object-needed-list=NEEDS_OBJECT_NAME.txt Name of file to which list of files that need object name is saved; default is NEEDS_OBJECT_NAME.txt

-p=NEEDS_POINTING_INFO.txt, --pointing-needed-list=NEEDS_POINTING_INFO.txt Name of file to which list of files that need pointing name is saved; default is NEEDS_POINTING_INFO.txt

-f=NEEDS_FILTER.txt, --filter-needed-list=NEEDS_FILTER.txt Name of file to which list of files that need filter is saved; default is NEEDS_FILTER.txt

-y=NEEDS_ASTROMETRY.txt, --astrometry-needed-list=NEEDS_ASTROMETRY.txt Name of file to which list of files that need astrometry is saved; default is NEEDS_ASTROMETRY.txt

DESCRIPTION

For each directory provided on the command line create a table in that directory with one row for each FITS file in the directory. The columns are FITS keywords extracted from the header of each file.

The list of default keywords extracted is available through the command line option `--list-default`.

Note: This feature is available only from the command line.

For more control over the parameters see `triage_fits_files()`

Note: This script is **NOT RECURSIVE**; it will not process files in subdirectories of the the directories supplied on the command line.

EXAMPLES

Invoking this script from the command line:

```
python run_triage.py /my/folder/of/images
```

Get list of default keywords included in summary table:

```
python run_triage.py --list-default
```

To work on the same folder from within python, do this:

```
from msumastro.scripts import run_triage
run_triage.main(['/my/folder/of/images'])
# or...
run_triage.main(['--list-default'])
```

Manual intervention: `quick_add_keys_to_file.py`

Warning: This script OVERWRITES the image files in the directories specified on the command line. There is NO WAY TO DISABLE this behavior.

Usage summary

```
usage: quick_add_keys_to_file.py [-h] [--file-list FILE_LIST]
                                 (--key-file KEY_FILE | --key-value KEY_VALUE KEY_VALUE)
                                 [files [files ...]]
```

Positional arguments:

files	Files in which to add/change keywords

Options:

--file-list	File with list of files in which keywords are to be changed
--key-file	File with keywords and values to be set
--key-value	Keyword to add/change

Add/modify keywords in FITS files.

DESCRIPTION

Add each of the keywords in either the `key_file` or specified on the command line to each of the files either listed in the file `file_list` or specified on the command line. If the keyword is already present its value is updated to the value in `key_file`. A HISTORY comment is added to the header for each keyword indicating which keyword was modified.

Warning: This script OVERWRITES the image files in the list specified on the command line. There is NO WAY to override this behavior.

Sorting files based on image properties

Note: By default this script makes a copy of the images being sorted. There is an option for moving the files instead.

This type of sorting is handy for working with images in GUI software like AstroImageJ or MaxImDL, but will make it harder to process the data programmatically in python.

Usage summary

```
usage: sort_files.py [-h] [-v] [-d DESTINATION_DIR] [--debug] [-n]
                     [--quiet-console] [--silent-console] [--move]
                     dir [dir ...]
```

Positional arguments:

dir Directory to process

Options:

-v=False, --verbose=False provide more information during processing

-d, --destination-dir Directory in which output from this script will be stored

--debug=False Turn on very detailed logging output

-n=False, --no-log-destination=False Do not write log files to destination directory

--quiet-console=False Log only errors (or worse) to console while running scripts

--silent-console=False Turn off all logging output to console

--move=False, -m=False Move files instead of copying them.

DESCRIPTION

For the directory provided on the command line sort the FITS files in this way:

```
    destination
        |
        |
        |
        |---'BIAS'
        |
        |---'DARK'
        |     |---exposure_time_1
        |     |---exposure_time_2, etc.
```

```
        |
        |---'FLAT'
        |       |---filter_1
        |       |     |---exposure_time_1
        |       |     |---exposure_time_2, etc.
        |       |
        |       |---filter_2, etc.
        |
        |---'LIGHT'
                |---object_1
                |       |---filter_1
                |       |     |---exposure_time_1
                |       |     |---exposure_time_2, etc.
                |       |
                |       |---filter_2, etc.
                |
                |---object_2
                |       |---filter_1
                |       |---filter_2, etc.
                |
                |---object_3, etc.
                |
                |---'no_object'
                        |---filter_1
                        |---filter_2, etc.
```

The names in single quotes, like *'bias'*, appear exactly as written in the directory tree created. Names like *exposure_time_1* are replaced with a value, for example 30.0 if the first dark exposure time is 30.0 seconds.

The directory `destination/calibration/flat/R` will contain all of the FITS files that are R-band flats.

Note: This script is **NOT RECURSIVE**; it will not process files in subdirectories of the the directories supplied on the command line.

Warning: Unless you explicitly supply a destination using the –destination-dir option the files will be copied/moved in the directory in which they currently exist. While this *should not* lead to information loss, since files are moved or copied but never deleted, you have been warned.

EXAMPLES

1.4 Manual header processing

1.4.1 Overview

Sometimes the standard data preparation will fail at one stage or another, most often because pointing information is missing for an image or because no object was found matching the RA/Dec of the image. Your tool of choice in such cases, either to add pointing information or to add object names is *quick_add_keys_to_file*. A broad discussion of using it is at *Fixes for cases that require intervention*.

This document provides some examples of using *quick_add_keys_to_file* from the command line. See the documentation for `add_keys()` for use from python scripts.

1.4.2 Examples

Command line only

Add the keyword "OBJECT", with value "EY UMa", to the file `image.fit`:

```
quick_add_keys_to_file.py --key-value object "EY UMa" image.fits
```

The same, but for all of the files that match the pattern `ey-uma*.fit`:

```
quick_add_keys_to_file.py --key-value object "EY UMa" ey-uma*.fits
```

The rest of the command line examples you have created a file called `keys.txt` with a list of keyword/value pairs and a list of files called `files.txt` (you can call the files whatever you want, of course)

Command line and supporting files

Format of the keyword file

A keyword file looks like this (you need the header line):

```
Keyword,Value
OBJECT,"EY UMa"
RA,"09:02:20.79"
DEC,"+49:49:09.7"
```

You can include as many keywords as you want, and they can have numerical values instead of string values in appropriate. If the value has two words, like the value for the keyword "OBJECT" above, it must be in quotes, like "EY UMa".

Keyword names are case insensitive because keywords in the FITS standard are case insensitive.

Format of the file list

A file list looks like this (yes, you need the header line):

```
File
MyFirstFile.fit
another_fits_file.fits
/or/even/the/full/path/to/a/fits/file.fit
```

Examples using keyword file/file list

Add all of the keywords in `keys.txt` to all of the files in `files.txt`:

```
quick_add_keys_to_file.py --key-file keys.txt --file-list files.txt
```

Add all of the keywords in `keys.txt` to the files `image1.fit` and `image2.fit`:

```
quick_add_keys_to_file.py --key-file keys.txt image1.fit image2.fit
```

Add keywords from the command line to all of the files in `files.txt`:

```
quick_add_keys_to_file.py --key-value my_key "some value" --file-list files.txt
```

1.5 Image Management

1.5.1 Working with a directory of images

For the sake of argument all of the examples below assume you are working in a directory that contains FITS images.

The class *ImageFileCollection* is meant to make working with a directory of FITS images easier by allowing you select the files you act on based on the values of FITS keywords in their headers.

It is initialized with the name of a directory containing FITS images and a list of FITS keywords you want the *ImageFileCollection* to be aware of. An example initialization looks like:

```
>>> from msumastro import ImageFileCollection
>>> keys = ['imagetyp', 'object', 'filter', 'exposure']
>>> ic1 = ImageFileCollection('.', keywords=keys) # only keep track of keys
```

You can use the wildcard * in place of a list to indicate you want the collection to use all keywords in the headers:

```
>>> ic_all = ImageFileCollection('.', keywords='*')
```

Most of the useful interaction with the image collection is via its `.summary` property, an **'~astropy.table.Table'_** of the value of each keyword for each file in the collection:

```
>>> ic1.summary.colnames
['imagetyp', 'object', 'filter']
>>> ic_all.summary.colnames
# long list of keyword names omitted
```

Selecting files

Selecting the files that match a set of criteria, for example all images in the I band with exposure time less than 60 seconds you could do:

```
>>> matches = (ic1.summary['filter'] == 'I' & ic1.summary['exposure'] < 60)
>>> my_files = summary['file'][matches]
```

The column `file` is added automatically when the image collection is created.

For more simple selection, when you just want files whose keywords exactly match particular values, say all 'I' band images with exposure time of 30 seconds, there is a convenience method `.files_filtered`:

```
>>> my_files = ic1.files_filtered(filter='I', exposure=30)
```

The optional arguments to `files_filtered` are used to filter the list of files.

Iterating over hdus, headers or data

Three methods are provided for iterating over the images in the collection, optionally filtered by keyword values.

For example, to iterate over all of the I band light images with exposure of 30 seconds, performing some basic operation on the data (very contrived example):

```
>>> for hdu in ic1.hdus(imagetyp='LiGhT', filter='I', exposure=30):
...     hdu.header['exposure']
...     new_data = hdu.data - hdu.data.mean()
```

Note that the names of the arguments to `hdus` here are the names of FITS keywords in the collection and the values are the values of those keywords you want to select. Note also that string comparisons are not case sensitive.

The other iterators are `headers` and `data`.

All of them have the option to also provide the file name in addition to the hdu (or header or data):

```
>>> for hdu, fname in ic1.hdus(return_fname=True,
...                            imagetyp='LiGhT', filter='I', exposure=30):
...     hdu.header['meansub'] = True
...     hdu.data = hdu.data - hdu.data.mean()
...     hdu.writeto(fname + '.new')
```

That last use case, doing something to several files and wanting to same them somewhere afterwards, is common enough that the iterators provide arguments to automate it.

Automatic saving from the iterators

There are three ways of triggering automatic saving.

1. One is with the argument `save_with_name`; it adds the value of the argument to the file name between the original base name and extension. The example below has (almsot) the same effect of the example above, subtracting the mean from each image and saving to a new file:

```
 >>> for hdu in ic1.hdus(save_with_name='_new',
 ...                     imagetyp='LiGhT', filter='I', exposure=30):
 ...     hdu.header['meansub'] = True
 ...     hdu.data = hdu.data - hdu.data.mean()

It saves, in the ``location`` of the image collection, a new FITS file with
the mean subtracted from the data, with ``_new`` added to the name; as an
example, if one of the files iterated over was ``intput001.fit`` then a new
file, in the same directory, called ``input001_new.fit`` would be created.
```

2. You can also provide the directory to which you want to save the files with `save_location`; note that you do not need to actually do anything to the hdu (or header or data) to cause the copy to be made. The example below copies all of the I-band light files with 30 second exposure from the original location to `"other_dir"`:

```
 >>> for hdu in ic1.hdus(save_location='other_dir',
 ...                     imagetyp='LiGhT', filter='I', exposure=30):
 ...     pass

This option can be combined with the previous one to also give the files a
new name.
```

3. Finally, if you want to live dangerously, you can overwrite the files in the location with the `overwrite` argument; use it carefully because it preserves no backup. The example below replaces each of the I-band light files with 30 second exposure with a file that has had the mean subtracted:

```
>>> for hdu in ic1.hdus(overwrite=True,
...                     imagetyp='LiGhT', filter='I', exposure=30):
...     hdu.header['meansub'] = True
...     hdu.data = hdu.data - hdu.data.mean()
```

msumastro.image_collection Module

ImageFileCollection(*arg, **kwd)

Classes

ImageFileCollection

class msumastro.image_collection.**ImageFileCollection**(**arg*, ***kwd*)

Bases: ccdproc.image_collection.ImageFileCollection

Attributes Summary

files	list of str, Unfiltered list of FITS files in location.
keywords	list of str, Keywords currently in the summary table.
location	str, Path name to directory containing FITS files
summary	
summary_info	Deprecated – use summary instead – astropy.table.Table of values of FITS keywords for files in the collection.

Methods Summary

data([do_not_scale_image_data])	Generator that yields each image in the collection.
files_filtered(**kwd)	Determine files whose keywords have listed values.
hdus([do_not_scale_image_data])	Generator that yields each HDU in the collection.
headers([do_not_scale_image_data])	Generator that yields each header in the collection.
refresh()	Refresh the collection by re-reading headers.
values(keyword[, unique])	List of values for a keyword.

Attributes Documentation

files

list of str, Unfiltered list of FITS files in location.

keywords

list of str, Keywords currently in the summary table.

Setting the keywords causes the summary table to be regenerated unless the new keywords are a subset of the old.

location

str, Path name to directory containing FITS files

summary

summary_info

Deprecated – use summary instead – astropy.table.Table of values of FITS keywords for files in the collection.

Each keyword is a column heading. In addition, there is a column called 'file' that contains the name of the FITS file. The directory is not included as part of that name.

Methods Documentation

data(*do_not_scale_image_data=False*, ***kwd*)

Generator that yields each image in the collection.

If any of the parameters `save_with_name`, `save_location` or `overwrite` evaluates to `True` the generator will write a copy of each FITS file it is iterating over. In other words, if `save_with_name` and/or `save_location` is a string with non-zero length, and/or `overwrite` is `True`, a copy of each FITS file will be made.

Parameters **save_with_name** : str

string added to end of file name (before extension) if FITS file should be saved after iteration. Unless `save_location` is set, files will be saved to location of the source files `self.location`

save_location : str

Directory in which to save FITS files; implies that FITS files will be saved. Note this provides an easy way to copy a directory of files–loop over the image with `save_location` set.

overwrite : bool

If `True`, overwrite input FITS files.

do_not_scale_image_data : bool

If `True`, prevents fits from scaling images. Default is `False`.

return_fname : bool, default is False

If True, return the tuple (header, file_name) instead of just header. The file name returned is the name of the file only, not the full path to the file.

kwd : dict

Any additional keywords are used to filter the items returned; see Examples for details.

Returns numpy.ndarray

If `return_fname` is `False`, yield the next image in the collection

(numpy.ndarray, str)

If `return_fname` is `True`, yield a tuple of (image, `file name`) for the next item in the collection.

files_filtered(***kwd*)

Determine files whose keywords have listed values.

`**kwd` is list of keywords and values the files must have.

If the keyword `include_path=True` is set, the returned list contains not just the filename, but the full path to each file.

The value '*' represents any value. A missing keyword is indicated by value ''

Example: >>> keys = ['imagetyp','filter'] >>> collection = ImageFileCollection('test/data', keywords=keys) >>> collection.files_filtered(imagetyp='LIGHT', filter='R') >>> collection.files_filtered(imagetyp='*', filter='')

NOTE: Value comparison is case *insensitive* for strings.

hdus (*do_not_scale_image_data=False*, ***kwd*)
Generator that yields each HDU in the collection.

If any of the parameters `save_with_name`, `save_location` or `overwrite` evaluates to `True` the generator will write a copy of each FITS file it is iterating over. In other words, if `save_with_name` and/or `save_location` is a string with non-zero length, and/or `overwrite` is `True`, a copy of each FITS file will be made.

Parameters **save_with_name** : str

string added to end of file name (before extension) if FITS file should be saved after iteration. Unless `save_location` is set, files will be saved to location of the source files `self.location`

save_location : str

Directory in which to save FITS files; implies that FITS files will be saved. Note this provides an easy way to copy a directory of files–loop over the HDU with `save_location` set.

overwrite : bool

If `True`, overwrite input FITS files.

do_not_scale_image_data : bool

If `True`, prevents fits from scaling images. Default is `False`.

return_fname : bool, default is False

If True, return the tuple (header, file_name) instead of just header. The file name returned is the name of the file only, not the full path to the file.

kwd : dict

Any additional keywords are used to filter the items returned; see Examples for details.

Returns astropy.io.fits.HDU

If `return_fname` is `False`, yield the next HDU in the collection

(astropy.io.fits.HDU, str)

If `return_fname` is `True`, yield a tuple of (HDU, `file name`) for the next item in the collection.

headers (*do_not_scale_image_data=True*, ***kwd*)
Generator that yields each header in the collection.

If any of the parameters `save_with_name`, `save_location` or `overwrite` evaluates to `True` the generator will write a copy of each FITS file it is iterating over. In other words, if `save_with_name` and/or `save_location` is a string with non-zero length, and/or `overwrite` is `True`, a copy of each FITS file will be made.

Parameters **save_with_name** : str

string added to end of file name (before extension) if FITS file should be saved after iteration. Unless `save_location` is set, files will be saved to location of the source files `self.location`

save_location : str

Directory in which to save FITS files; implies that FITS files will be saved. Note this provides an easy way to copy a directory of files–loop over the header with `save_location` set.

overwrite : bool

If `True`, overwrite input FITS files.

do_not_scale_image_data : bool

If `True`, prevents fits from scaling images. Default is `True`.

return_fname : bool, default is False

If True, return the tuple (header, file_name) instead of just header. The file name returned is the name of the file only, not the full path to the file.

kwd : dict

Any additional keywords are used to filter the items returned; see Examples for details.

Returns astropy.io.fits.Header

If `return_fname` is `False`, yield the next header in the collection

(astropy.io.fits.Header, str)

If `return_fname` is `True`, yield a tuple of (header, `file name`) for the next item in the collection.

refresh()
Refresh the collection by re-reading headers.

values(*keyword*, *unique=False*)
List of values for a keyword.

Parameters **keyword** : str

Keyword (i.e. table column) for which values are desired.

unique : bool, optional

If True, return only the unique values for the keyword.

Returns list

Values as a list.

1.5.2 Turning an image collection into a tree

The class *TableTree* turns an Astropy Table into a tree based on the values in a particular column or columns.

TableTree(*args, **kwd)	Base class for grouping images hierarchically into a tree based on metadata.
RecursiveTree()	A dict-base recursive tree.

TableTree

class msumastro.table_tree.**TableTree**(**args*, ***kwd*)
Bases: *msumastro.table_tree.RecursiveTree*

Base class for grouping images hierarchically into a tree based on metadata.

Parameters **table** : astropy.table.Table instance

Table containing the metadata to be used for grouping images.

tree_keys : list of str

Keys to be used in grouping images. Each key must be the name of a column in *table*.

index_key : str

Key which is used to indicate which rows of the input table are in each group; it must be the name of one of the columns in *table*. Values of the index must uniquely identify rows of the table (in database parlance, index must be able to serve as a primary key for the table).

Raises **TypeError**

Raised if the number of initialization arguments is incorrect or the types of any of the arguments is incorrect.

Attributes

table	astropy.table.Table of metadata used to group rows.
tree_keys	list of str, Table columns to be used in grouping the rows.
index_key	str, Name of column whose values uniquely identify each row.

Attributes Summary

index_key	str, Name of column whose values uniquely identify each row.
table	astropy.table.Table of metadata used to group rows.
tree_keys	list of str, Table columns to be used in grouping the rows.

Methods Summary

walk(*args, **kwd)	Walk the grouped tree

Attributes Documentation

index_key
str, Name of column whose values uniquely identify each row.

table
astropy.table.Table of metadata used to group rows.

tree_keys
list of str, Table columns to be used in grouping the rows.

Methods Documentation

walk(**args*, ***kwd*)
Walk the grouped tree

The functionality provided is similar to that in os.walk: starting at the top of tree, yield a tuple of return values indicating parents, children and rows at each level of the tree.

Parameters **None**

Returns **parents, children, index** : lists

parents : list

Dictionary keys that led to this return

children : list

Child nodes at this level

index : list

Index values for the items in the table that correspond to the values in *parents*

RecursiveTree

class msumastro.table_tree.**RecursiveTree**

Bases: collections.OrderedDict

A dict-base recursive tree.

Methods Summary

add_keys(keys[, value])	

Methods Documentation

add_keys(*keys*, *value=None*)

1.6 Header processing

1.6.1 Introduction

The *msumastro.header_processing* subpackage contains classes and functions that do the work of modifying headers. There are several scripts for *Automated header processing* and *Manual header processing* provided to carry out the most common types of headers processing.

1.6.2 Reference/API

msumastro.header_processing.fitskeyword Module

Classes

FITSKeyword([name, value, comment, synonyms])	Represents a FITS keyword, which may have several synonyms.

FITSKeyword

class msumastro.header_processing.fitskeyword.**FITSKeyword**(*name=None*, *value=None*, *comment=None*, *synonyms=None*)

Bases: object

Represents a FITS keyword, which may have several synonyms.

Parameters **name** : str, optional

Name of the keyword; case insensitive

value : str or numeric type, optional

Value of the keyword; this class imposes no constraints on the type of the keyword but if you intend to save the value in a FITS header you should be aware of the restrictions the FITS standard places on keyword values.

comment : str, optional

Description of the keyword.

synonyms : str or list of str, optional

Synonyms for this keyword. Synonyms are to look for a value in a FITS header and to set multiple keywords to the same value in a FITS header.

Attributes Summary

name	Primary name of the keyword.
names	All names, including synonyms, for this keyword, as a list.
synonyms	List of synonyms for the keyword.

Methods Summary

add_to_header(hdu_or_header[, ...])	Add keyword to FITS header.
history_comment([with_name])	Produce a string describing changes to the keyword value.
set_value_from_header(hdu_or_header)	Set value of keyword from FITS header.

Attributes Documentation

name
Primary name of the keyword.

names
All names, including synonyms, for this keyword, as a list.

synonyms
List of synonyms for the keyword.

Methods Documentation

add_to_header(*hdu_or_header*, *with_synonyms=True*, *history=False*)
Add keyword to FITS header.

Parameters **hdu_or_header** : astropy.io.fits.Header or astropy.io.fits.PrimaryHDU

Header/HDU to which the keyword is to be added.

with_synonyms : bool, optional

Control whether a keyword is added for each of the synonyms for the keyword. Default is True.

history : bool, optional

Control whether a history comment is added to the header; if True a history comment is added for *each* of the keyword names added to the header, including synonyms.

history_comment(*with_name=None*)

Produce a string describing changes to the keyword value.

Parameters **with_name** : str, optional

Name to use for the keyword in the history comment. Default is the *name* attribute of the *Keyword*.

set_value_from_header(*hdu_or_header*)

Set value of keyword from FITS header.

Values are obtained from the header by looking for the keyword by its primary name and any synonyms. If multiple values are found they are checked for consistency.

Parameters **hdu_or_header: astropy.io.fits.Header or astrop.io.fits.PrimaryHDU**

Header from which the keyword value should be taken.

Raises **ValueError**

If *hdu_or_header* is of the wrong type, or the keyword (or synonyms) are not found in the header, or multiple non-identical values are found.

msumastro.header_processing.astrometry Module

Functions

call_astrometry(filename[, sextractor, ...])	Wrapper around astrometry.net solve-field.
add_astrometry(filename[, overwrite, ...])	Add WCS headers to FITS file using astrometry.net

call_astrometry

msumastro.header_processing.astrometry.**call_astrometry**(*filename*, *sextractor=False*, *feder_settings=True*, *no_plots=True*, *minimal_output=True*, *save_wcs=False*, *verify=None*, *ra_dec=None*, *overwrite=False*, *wcs_reference_image_center=True*)

Wrapper around astrometry.net solve-field.

Parameters **sextractor** : bool or str, optional

`True` to use *sextractor*, or a `str` with the path to sextractor.

feder_settings : bool, optional

Set True if you want to use plate scale appropriate for Feder Observatory Apogee Alta U9 camera.

no_plots : bool, optional

`True` to suppress astrometry.net generation of plots (pngs showing object location and more)

minimal_output : bool, optional

If `True`, suppress, as separate files, output of: WCS header, RA/Dec object list, matching objects list, but see also *save_wcs*

save_wcs : bool, optional

If `True`, save WCS header even if other output is suppressed with *minimial_output*

verify : str, optional

Name of a WCS header to be used as a first guess for the astrometry fit; if this plate solution does not work the solution is found as though *verify* had not been specified.

ra_dec : list or tuple of float

(RA, Dec); also limits search radius to 1 degree.

overwrite : bool, optional

If `True`, perform astrometry even if astrometry.net files from a previous run are present.

wcs_reference_image_center :

If `True`, force the WCS reference point in the image to be the image center.

add_astrometry

`msumastro.header_processing.astrometry.`**`add_astrometry`**(*filename*, *overwrite=False*, *ra_dec=None*, *note_failure=False*, *save_wcs=False*, *verify=None*, *try_builtin_source_finder=False*)

Add WCS headers to FITS file using astrometry.net

Parameters **overwrite** : bool, optional

Set `True` to overwrite the original file. If *False*, the file astrometry.net generates is kept.

ra_dec : list or tuple of float or str

(RA, Dec) of field center as either decimal or sexagesimal; also limits search radius to 1 degree.

note_failure : bool, optional

If `True`, create a file with extension "failed" if astrometry.net fails. The "failed" file contains the error messages genreated by astrometry.net.

try_builtin_source_finder : bool

If true, try using astrometry.net's built-in source extractor if sextractor fails.

save_wcs :

verify :

See *call_astrometry()*

Returns bool

`True` on success.

Notes

Tries a couple strategies before giving up: first sextractor, then, if that fails, astrometry.net's built-in source extractor.

It also cleans up after astrometry.net, keeping only the new FITS file it generates, the .solved file, and, if desired, a ".failed" file for fields which it fails to solve.

For more flexible invocation of astrometry.net, see *call_astrometry()*

msumastro.header_processing.feder Module

Classes

FederSite	The Feder Observatory site.
ImageSoftware(name[, fits_name, ...])	Represents software that takes images at telescope.
Instrument(name[, fits_names, rows, ...])	Telescope instrument with simple properties.
ApogeeAltaU9()	The Apogee Alta U9
MaximDL4()	Represents MaximDL version 4, all sub-versions
MaximDL5()	Represents MaximDL version 5, all sub-versions.

FederSite

class msumastro.header_processing.feder.**FederSite**

Bases: astropy.coordinates.earth.EarthLocation

The Feder Observatory site.

An astropy location with the observatory location pre-set to:

- *lat* = 46.86678 degrees North
- *long* = -96.453278 degrees East
- *height* = 311.8 meters

and a few additional properties/methods that are convenient:

- *name* = Feder Observatory

Attributes Summary

name

Attributes Documentation

name

ImageSoftware

class msumastro.header_processing.feder.**ImageSoftware**(*name*, *fits_name=None*, *major_version=None*, *minor_version=None*, *bad_keywords=None*, *fits_keyword=None*, *purged_flag_keyword=None*)

Bases: object

Represents software that takes images at telescope.

Parameters **name** : str

Name of the software. Can be the same is the name in the FITS file, or not.

fits_keyword : str

Name of the FITS keyword that contains the name of the software.

fits_name : list of str

Name of the software as written in the FITS file

major_version : int

Major version number of the software.

minor_version : int

Minor version number of the software.

bad_keywords : list of strings

Names of any keywords that should be removed from the FITS before further processing.

purged_flag_keyword : str, optional

Name of the keyword which indicates whether bad keywords have already been purged. Default value is 'PURGED'

Methods Summary

created_this(version_string)	Indicate whether version string matches this software

Methods Documentation

created_this(*version_string*)

Indicate whether version string matches this software

Parameters **version_string** : str

String from FITS header that indicates software version.

Returns bool

True if the version string matches the software instance.

Instrument

class msumastro.header_processing.feder.**Instrument**(*name*, *fits_names=None*, *rows=0*, *columns=0*, *image_unit=None*, *trim_region=None*, *useful_overscan_region=None*)

Bases: object

Telescope instrument with simple properties.

Parameters **name** : str

Name of the instrument.

fits_names : list of str

List of names by which the instrument is known in FITS headers

rows : int

Number of rows in an image produced by this instrument, including overscan.

columns : int

Number of columns in an image produced by this instrument, including overscan.

image_unit : astropy.units.Unit

Unit of the image; default value is None

trim_region : string

Region of the CCD that should be preserved after overscan subtraction. Should use *FITS* conventions for specifying slices (i.e. slice starts at 1, includes endpoint, and uses FITS NAXIS1, NAXIS2 for order of indices).

useful_overscan_region : string

Complete specification of the region of the CCD actually useful for overscan calibration. This may (or may not) be smaller than the entire portion of the chip the manufacturer labels as overscan. Should use *FITS* conventions for specifying slices (i.e. slice starts at 1, includes endpoint, and uses FITS NAXIS1, NAXIS2 for order of indices).

Examples

Consider an image whose dimensions as given in its FITS header are NAXIS1 = 3085 and NAXIS2 = 2048 with an overscan region that begins at position 3073 along axis 1. The useful part of that overscan is from FITS column 3076 up to and including, 3079, and the full range of rows (NAXIS2). The correct overscan settings for this instrument are:

```
# Note not all of the overscan region is actually useful.
useful_overscan_region = '[3076:3079, :]'
# But the whole overscan region should be trimmed away in reduction.
trim_region = '[1:3073, :]'
```

Methods Summary

has_overscan(image_dimensions)	Determine whether an image taken by this instrument has overscan

Methods Documentation

has_overscan(*image_dimensions*)

Determine whether an image taken by this instrument has overscan

Parameters **image_dimensions** : list-like with two elements

Shape of the image; can be any type as long as it has two elements. The order should be the FITS order, `NAXIS1` then `NAXIS2`.

Returns bool

Indicates whether or not image has overscan present.

ApogeeAltaU9

class msumastro.header_processing.feder.**ApogeeAltaU9**

Bases: *msumastro.header_processing.feder.Instrument*

The Apogee Alta U9

MaximDL4

class msumastro.header_processing.feder.**MaximDL4**

Bases: *msumastro.header_processing.feder.ImageSoftware*

Represents MaximDL version 4, all sub-versions

MaximDL5

class msumastro.header_processing.feder.**MaximDL5**

Bases: *msumastro.header_processing.feder.ImageSoftware*

Represents MaximDL version 5, all sub-versions.

Subversions are included by listing the FITS names of all versions that have been used at Feder Observatory.

msumastro.header_processing.patchers Module

Functions

IRAF_image_type(image_type)	Convert MaximDL default image type names to IRAF
add_image_unit(header[, history])	Add unit of image to header.
add_object_info([directory, object_list, ...])	Add object information to FITS files that contain pointing information give
add_object_pos_airmass(header[, history])	Add object information, such as RA/Dec and airmass.
add_overscan_header(header[, history])	Add overscan information to a FITS header.
add_ra_dec_from_object_name([directory, ...])	Add RA/Dec to FITS file that has object name but no pointing.
add_time_info(header[, history])	Add JD, MJD, LST to FITS header
change_imagetype_to_IRAF(header[, history])	Change IMAGETYP to default used by IRAF
get_software_name(header[, file_name, ...])	Determine the name of the software that created FITIS header
history(function[, mode, time])	Construct nicely formatted start/end markers in FITS history.
list_name_is_url(name)	
patch_headers([dir, new_file_ext, ...])	Add minimal information to Feder FITS headers.
purge_bad_keywords(header[, history, force, ...])	Remove keywords from FITS header that may be incorrect
read_object_list([directory, input_list, ...])	Read a list of objects from a text file.

IRAF_image_type
msumastro.header_processing.patchers.**IRAF_image_type**(*image_type*)
Convert MaximDL default image type names to IRAF

Parameters **image_type** : str

Value of the FITS header keyword IMAGETYP; acceptable values are below in Notes.

Returns str

IRAF image type (one of 'BIAS', 'DARK', 'FLAT' or 'LIGHT')

Notes

The MaximDL default is, e.g. 'Bias Frame', which IRAF calls 'BIAS'. Can safely be called with an IRAF-style image_type.

add_image_unit
msumastro.header_processing.patchers.**add_image_unit**(*header*, *history=True*)
Add unit of image to header.

Parameters **header** : astropy.io.fits.Header

Header object in which image type is to be changed.

history : bool, optional

If *True*, add history of keyword modification to *header*.

add_object_info
msumastro.header_processing.patchers.**add_object_info**(*directory=None*, *object_list=None*, *object_list_dir=None*, *match_radius=20.0*, *new_file_ext=None*, *save_location=None*, *overwrite=False*, *detailed_history=True*)
Add object information to FITS files that contain pointing information given a list of objects.

Parameters **directory** : str

Directory containing the FITS files to be fixed. Default is the current directory, `.`.

object_list : str, optional

Name of file containing list of objects. Default is set by *read_object_list()* which also explains the format of this file.

object_list_dir : str, optional

Directory in which the *object_list* is contained. Default is *directory*.

match_radius : float, optional

Maximum distance, in arcmin, between the RA/Dec of the image and a particular object for the image to be considered an image of that object.

new_file_ext : str, optional

Name added to the FITS files with updated header information. It is added to the base name of the input file, between the old file name and the *.fit* or *.fits* extension. Default is 'new'.

save_location : str, optional

Directory to which the patched files should be written, if not *dir*.

overwrite : bool, optional

Set to *True* to replace the original files.

add_object_pos_airmass

msumastro.header_processing.patchers.**add_object_pos_airmass**(*header*, *history=False*)

Add object information, such as RA/Dec and airmass.

Parameters **header** : astropy..io.fits.Header

FITS header to be modified.

history : bool

If *True*, write history for each keyword changed.

Notes

Has side effect of setting feder site JD to JD-OBS, which means it also assume JD.value has been set.

add_overscan_header

msumastro.header_processing.patchers.**add_overscan_header**(*header*, *history=True*)

Add overscan information to a FITS header.

Parameters **header** : astropy.io.fits.Header

Header object to which overscan is to be added.

history : bool, optional

If *True*, add history of keyword modification to *header*.

Returns list of str

List of the keywords added to the header by this function.

add_ra_dec_from_object_name

msumastro.header_processing.patchers.**add_ra_dec_from_object_name**(*directory=None*, *new_file_ext=None*, *object_list=None*, *object_list_dir=None*)

Add RA/Dec to FITS file that has object name but no pointing.

Parameters **dir** : str, optional

Directory containing the files to be patched. Default is the current directory, **.**

new_file_ext : str, optional

Name added to the FITS files with updated header information. It is added to the base name of the input file, between the old file name and the *.fit* or *.fits* extension. Default is 'new'.

object_list : str, optional

Name of file containing list of objects. Default is set by `read_object_list()` which also explains the format of this file.

object_list_dir : str, optional

Directory in which the *object_list* is contained. Default is *directory*.

add_time_info

msumastro.header_processing.patchers.**add_time_info**(*header*, *history=False*)

Add JD, MJD, LST to FITS header

Parameters **header** : astropy..io.fits.Header

FITS header to be modified.

history : bool

If *True*, write history for each keyword changed.

change_imagetype_to_IRAF

msumastro.header_processing.patchers.**change_imagetype_to_IRAF**(*header*, *history=True*)

Change IMAGETYP to default used by IRAF

Parameters **header** : astropy.io.fits.Header

Header object in which image type is to be changed.

history : bool, optional

If *True*, add history of keyword modification to *header*.

get_software_name

msumastro.header_processing.patchers.**get_software_name**(*header*, *file_name=None*, *use_observatory=None*)

Determine the name of the software that created FITIS header

Parameters **header** : astropy.io.fits Header

Header from a FITS extension/hdu

file_name : str, optional

Name of the file containing this header; used to add information to error/warning messages.

use_observatory : msumastro.Feder instance, optional

Object that contains names of FITS keywords that might be present and contain name of the software that made this header. The default value is the instance defined at the beginning of this module

Returns msumastro.feder.Software object

history
msumastro.header_processing.patchers.**history**(*function*, *mode=None*, *time=None*)

Construct nicely formatted start/end markers in FITS history.

Parameters **function** : func

Function calling *history*

mode : str, 'begin' or 'end'

A different string is produced for the beginning and the end. Default is 'begin'.

time : datetime

If not set, defaults to current date/time.

list_name_is_url
msumastro.header_processing.patchers.**list_name_is_url**(*name*)

patch_headers
msumastro.header_processing.patchers.**patch_headers**(*dir=None*, *new_file_ext=None*, *save_location=None*, *overwrite=False*, *purge_bad=True*, *add_time=True*, *add_apparent_pos=True*, *add_overscan=True*, *fix_imagetype=True*, *add_unit=True*)

Add minimal information to Feder FITS headers.

Parameters **dir** : str, optional

Directory containing the files to be patched. Default is the current directory, `.`

new_file_ext : str, optional

Name added to the FITS files with updated header information. It is added to the base name of the input file, between the old file name and the *.fit* or *.fits* extension. Default is 'new'.

save_location : str, optional

Directory to which the patched files should be written, if not *dir*.

overwrite : bool, optional

Set to *True* to replace the original files.

purge_bad : bool, optional

Remove "bad" keywords form header before any other processing. See *purge_bad_keywords()* for details.

add_time : bool, optional

If `True`, add time information (e.g. JD, LST); see *add_time_info()* for details.

add_apparent_pos : bool, optional

If `True`, add apparent position (e.g. alt/az) to headers. See *add_object_pos_airmass()* for details.

add_overscan : bool, optional

If `True`, add overscan keywords to the headers. See *add_overscan_header()* for details.

fix_imagetype : bool, optional

If `True`, change image types to IRAF-style. See *change_imagetype_to_IRAF()* for details.

add_unit : bool, optional

If `True`, add image unit to FITS header.

purge_bad_keywords

msumastro.header_processing.patchers.**purge_bad_keywords**(*header*, *history=False*, *force=False*, *file_name=None*)

Remove keywords from FITS header that may be incorrect

Parameters **header** : astropy.io.fits.Header

Header from which the bad keywords (as defined by the software that recorded the image) should be purged.

history : bool

If *True* write detailed history for each keyword removed.

force : bool

If *True*, force keywords to be purged even if the FITS header indicates it has already been purged.

file_name : str, optional

Name of file containing the header; if provided it is used to generate more informative log messages.

read_object_list

msumastro.header_processing.patchers.**read_object_list**(*directory=None*, *input_list=None*, *skip_consistency_check=False*, *check_radius=20.0*, *skip_lookup_from_object_name=False*)

Read a list of objects from a text file.

Parameters **directory** : str

Directory containing the file. Default is the current directory, `.`

input_list : str, optional

Name of the file or URL of file. Default value is `obsinfo.txt`. If the name is a URL the directory argument is ignored.

skip_consistency_check : bool optional

If `True`, skip checking whether objects on the list have unique coordinates given *check_radius*.

check_radius : float, optional

Match radius, in arcminutes. Objects on the list must be separated by an angular distance greater than this for the list to be self-consistent.

skip_lookup_from_object_name : bool, optional

Set to `True` to skip lookup of coordinates from Simbad if RA/Dec are not in the object file.

Notes

There are two file formats; one contains just a list of objects, the other has an RA and Dec for each object.

In both types any lines that start with # are ignored and treated as comments.

File with list of objects only:

- Object coordinates are determined by lookup with Simbad. You should make sure the object names you use are known to simbad.
- The first non-comment line MUST be the word `object` and only the word `object`. It is case sensitive; `Object` or `OBJECT` will not work.
- Remaining line(s) are name(s) of object(s), one per line. Case does **not** matter for object name.
- Example:

```
# my list is below
object
m101
sz lyn
# the next object is after this comment
RR LYR
```

File with object name and position:

- RA and Dec **must be J2000**.
- RA **must be given in hours**, though it can be either sexagesimal (e.g. `19:25:27.9`) or decimal (e.g. `19.423861`).
- Dec **must be given in degrees**, though it can be either sexagesimal (e.g. `42:47:3.69`) or decimal (e.g. `42.7843583`)
- The first non-comment line MUST be these words: `object,RA,Dec`. These are column headings for your file. It is **not** case sensitive; for example, using `DEC` instead of `Dec` will work.
- Each remaining line should be an object name, object RA and Dec. Case does **not** matter for object name.
- Example:

```
# my list with RA and Dec
# RA and Dec are assumed to be J2000
# RA MUST BE IN HOURS
# DEC MUST BE IN DEGREES
object,RA,Dec
m101,14:03:12.583, +54:20:55.50
# note that the leading sign for the Dec is optional if Dec is
# positive
sz lyn,08:09:35.748, 44:28:17.61
# You can mix sexagesimal and decimal RA/Dec.
RR Lyr, 19.423861,42.7843583
```

1.7 Index

- genindex
- modindex

m

msumastro.header_processing.astrometry, 25
msumastro.header_processing.feder, 27
msumastro.header_processing.fitskeyword, 23
msumastro.header_processing.patchers, 30
msumastro.image_collection, 18
msumastro.scripts.quick_add_keys_to_file, 12
msumastro.scripts.run_astrometry, 10
msumastro.scripts.run_patch, 8
msumastro.scripts.run_standard_header_process, 8
msumastro.scripts.run_triage, 11
msumastro.scripts.sort_files, 13
msumastro.table_tree, 21

A

add_astrometry() (in module msumastro.header_processing.astrometry), 26
add_image_unit() (in module msumastro.header_processing.patchers), 31
add_keys() (msumastro.table_tree.RecursiveTree method), 23
add_object_info() (in module msumastro.header_processing.patchers), 31
add_object_pos_airmass() (in module msumastro.header_processing.patchers), 32
add_overscan_header() (in module msumastro.header_processing.patchers), 32
add_ra_dec_from_object_name() (in module msumastro.header_processing.patchers), 32
add_time_info() (in module msumastro.header_processing.patchers), 33
add_to_header() (msumastro.header_processing.fitskeyword.FITSKeyword method), 24
ApogeeAltaU9 (class in msumastro.header_processing.feder), 30

C

call_astrometry() (in module msumastro.header_processing.astrometry), 25
change_imagetype_to_IRAF() (in module msumastro.header_processing.patchers), 33
created_this() (msumastro.header_processing.feder.ImageSoftware method), 28

D

data() (msumastro.image_collection.ImageFileCollection method), 19

F

FederSite (class in msumastro.header_processing.feder), 27
files (msumastro.image_collection.ImageFileCollection attribute), 18
files_filtered() (msumastro.image_collection.ImageFileCollection method), 19
FITSKeyword (class in msumastro.header_processing.fitskeyword), 23

G

get_software_name() (in module msumastro.header_processing.patchers), 33

H

has_overscan() (msumastro.header_processing.feder.Instrument method), 30
hdus() (msumastro.image_collection.ImageFileCollection method), 19
headers() (msumastro.image_collection.ImageFileCollection method), 20
history() (in module msumastro.header_processing.patchers), 34
history_comment() (msumastro.header_processing.fitskeyword.FITSKeyword method), 25

I

ImageFileCollection (class in msumastro.image_collection), 18
ImageSoftware (class in msumastro.header_processing.feder), 27
index_key (msumastro.table_tree.TableTree attribute), 22
Instrument (class in msumastro.header_processing.feder), 28
IRAF_image_type() (in module msumastro.header_processing.patchers), 30

K

keywords (msumastro.image_collection.ImageFileCollection attribute), 18

L

list_name_is_url() (in module msumastro.header_processing.patchers), 34
location (msumastro.image_collection.ImageFileCollection attribute), 18

M

MaximDL4 (class in msumastro.header_processing.feder), 30
MaximDL5 (class in msumastro.header_processing.feder), 30
msumastro.header_processing.astrometry (module), 25
msumastro.header_processing.feder (module), 27
msumastro.header_processing.fitskeyword (module), 23
msumastro.header_processing.patchers (module), 30
msumastro.image_collection (module), 18
msumastro.scripts.quick_add_keys_to_file (module), 12
msumastro.scripts.run_astrometry (module), 10
msumastro.scripts.run_patch (module), 8
msumastro.scripts.run_standard_header_process (module), 8
msumastro.scripts.run_triage (module), 11
msumastro.scripts.sort_files (module), 13
msumastro.table_tree (module), 21

N

name (msumastro.header_processing.feder.FederSite attribute), 27
name (msumastro.header_processing.fitskeyword.FITSKeyword attribute), 24
names (msumastro.header_processing.fitskeyword.FITSKeyword attribute), 24

P

patch_headers() (in module msumastro.header_processing.patchers), 34
purge_bad_keywords() (in module msumastro.header_processing.patchers), 35

R

read_object_list() (in module msumastro.header_processing.patchers), 35
RecursiveTree (class in msumastro.table_tree), 23
refresh() (msumastro.image_collection.ImageFileCollection method), 21

S

set_value_from_header() (msumastro.header_processing.fitskeyword.FITSKeyword method), 25
summary (msumastro.image_collection.ImageFileCollection attribute), 18
summary_info (msumastro.image_collection.ImageFileCollection attribute), 18
synonyms (msumastro.header_processing.fitskeyword.FITSKeyword attribute), 24

T

table (msumastro.table_tree.TableTree attribute), 22
TableTree (class in msumastro.table_tree), 21
tree_keys (msumastro.table_tree.TableTree attribute), 22

V

values() (msumastro.image_collection.ImageFileCollection method), 21

W

walk() (msumastro.table_tree.TableTree method), 22