DrizzlePac Documentation

Release 0.1.dev50+g6b69dc9.d20240423

Warren Hack, Nadia Dencheva, Chris Sontag, Megan Sosey, Michael Droettboom, Mihai Cara

Apr 23, 2024

CONTENTS

DrizzlePac is a set of software tools maintained by the Space Telescope Science Institute (STScI) that is designed to align and combine astronomical images obtained with the Hubble Space Telescope. DrizzlePac includes core features such as AstroDrizzle, Tweakreg, and the Hubble Archival Program (HAP) data processing pipeline that creates Single-Visit and Multi-Visit Mosaics (SVMs and MVMs). This readthedocs provides resources for users to run DrizzlePac in order to create custom image mosaics, and documents the procedures used to produce the MAST SVM and MVM data products. Please see the Getting Started guide and DrizzlePac website for additional information.

CHAPTER

ONE

GETTING STARTED

DrizzlePac is designed to optimally align and combine multiple exposures into a single "drizzled" image in order to recover the maximum possible sensitivity and spatial resolution from HST imaging. STScI uses DrizzlePac to produce science-ready mosaics that can be accessed through the Mikulski Archive for Space Telescopes (MAST). However, the pipeline cannot optimize the DrizzlePac settings for every observation type and science case, so users are encouraged to create their own mosaics to incorporate custom calibrations, alignment strategies, and fine-tuning of the AstroDrizzle parameters. This Getting Started guide provides instructions on how to install DrizzlePac, as well as links to the main website, handbook, example notebooks, and instructions for requesting assistance through the HST Help Desk.

1.1 Installation

1.1.1 Conda (Recommended)

`Drizzlepac` is installed when you install the `stenv` conda environment (a replacement for `astroconda`). Select your desired release and follow the instructions on the installation page.

1.1.2 Install with pip

```
pip install git+https://github.com/spacetelescope/drizzlepac.git
```

The option `--no-use-pep517` MAY be required in order to correctly build the C extensions with `pip` versions up to 22.2, after commenting out the `build-backend` from the `pyproject.toml` config file.

Support for installing using `pip` is still evolving, so use of this command is provided on an experimental basis for now.

1.1.3 From Source

Clone this repository

```
git clone https://github.com/spacetelescope/drizzlepac
cd drizzlepac
```

Build the Documentation

Note: If you intend to use `drizzlepac`'s embedded help feature from within an interactive `python` or `ipython` session, we recommend you do not skip this step.

```
cd doc/
make html
```

Install drizzlepac

```
python setup.py install
```

1.2 Drizzlepac Resources

Here we provide a list of DrizzlePac related pages for users, including links to the main DrizzlePac webpage and handbook, example workflow notebooks, and resources for troubleshooting issues that may arise. If your issues are not covered or you have additional questions, please contact the HST Help Desk at the link below.

1.3 Contributor Resources

1.3.1 Basic Contribution Instructions

`drizzlepac` is an open source python package, and the source code is available in the DrizzlePac Github repository. New contributions and contributors are very welcome! Do not hesitate to reach out to the package maintainers if you are new to open-source development or if you have any questions/concerns. We just ask that all contributors adhere to the Space Telescope Code of Conduct.

Reporting bugs / requesting a new feature

If you would like to report a bug or request a new feature, this can be done by opening a new issue.

Contributing code

If you would like to contribute code, this is done by submitting a pull request to the "main" branch of `spacetelescope/drizzlepac`. To do this, we recommend the following workflow (which assumes you already have a Github account / command line tools). If you are also new to git, please refer to the git reference manual for an overview of git basics.

Step 1: Forking and cloning the `drizzlepac` repository

First, to clarify some terms that will be commonly used here:

- `upstream` refers to the main `drizzlepac` repository. this is where code from all contributors is ultimately merged into and where releases of the package will be made from.
- `origin` refers to the online fork you made of the upstream `drizzlepac` repository.
- `local` refers to the clone you made of the origin on your computer.
- The term `remote` in this context can refer to origin, or upstream. in general, it means anything hosted online on Github.

The first step is to create your own 'remote' (online) and 'local' (on your machine) clones of the central `spacetelescope/drizzlepac` repository. You will make code changes on your machine to your 'local' clone, push these to 'origin' (your online fork), and finally, open a pull request to the ''main" branch of `spacetelescope/drizzlepac`.

1. On the 'spacetelescope/drizzlepac' Github repository page, 'fork' the DrizzlePac repository to your own account space by clicking the appropriate button on the upper right-hand side. This will create an online clone of the main DrizzlePac repository but instead of being under the 'spacetelescope' organization, it will be under your personal account space. It is necessary to fork the repository so that many contributors aren't making branches directly on 'spacetelescope/drizzlepac'.
2. Now that you have remotely forked `drizzlepac`, it needs to be downloaded to your machine. To create this 'local' clone, choose an area on your file system and use the `git clone` command to dowload your remote fork on to your machine.

If you are using an SSH key to authenticate.

```
>> cd directory
>> git clone git@github.com:<your_username>/drizzlepac.git
```

Otherwise use the following.

```
>> cd directory
>> git clone https://github.com/<your_username>/drizzlepac.git
```

3. Make sure that your references to 'origin' and 'upstream' are set correctly - you will need this to keep everything in sync and push your changes online. While your inital local clone will be an exact copy of your remote, which is an exact copy of the 'upstream' `spacetelescope/drizzlepac`, these all must be kept in sync manually (via git fetch/pull/push).

   To check the current status of these references:

```
>> git remote -v
```

After your inital clone, you will likely be missing the reference to 'upstream' (which is just the most commonly used name in git to refer to the main project repository - you can call this whatever you want but the origin/upstream conventions are most commonly used) - to set this, use the `add` git command:

If you are using an SSH key to authenticate.

```
>> git remote add upstream git@github.com:spacetelescope/drizzlepac.git
```

Otherwise, you can simply set it to the repository URL but you will have to enter your password every time you fetch/push

```
>> git remote add upstream https://github.com/spacetelescope/drizzlepac.git
```

If you ever want to reset these URLs, add references to other remote forks of `drizzlepac` for collaboration, or change from URL to SSH, you can use the related `git remote set-url` command.

Step 2: Creating a branch for your changes

It is a standard practice in git to create a new 'branch' (off `upstream/main`) for each new feature or bug fix. You can call this branch whatever you like - in this example, we'll call it 'my_feature'. First, make sure you have all recent changes to upstream by 'fetching' them:

```
>> git fetch upstream
```

The following will create a new branch off local/main called 'my_feature', and automatically switch you over to your new branch.

```
>> git checkout -b my_feature upstream/main
```

Step 3: Installing `drizzlepac` for development

Next, you will create a new Python environment where you will install your new branch of `drizzlepac`. We will show here how to use conda to manage environments, but there are other options.

1. Create a conda environment.

It is good practice to maintain different environments for different versions of DrizzlePac and its dependencies. You will likely want to maintain one, for example, for the latest released version of DrizzlePac (i.e. what you get by doing `pip install drizzlepac`), as well as one for development. Assuming the user has conda

installed, here we will create a new conda environment called 'drizzlepac_dev' where we can install the new branch of our cloned repository.

```
>> conda create -n drizzlepac_dev python
```

Doing this will create a new environment with just some basic packages (i.e setuptools, pip) installed.

2. Installing `drizzlepac` in this environment

Make sure you are in your new environment:

```
>> conda activate drizzlepac_dev
```

And now in the top level of your local `drizzlepac` repository, ensuring you're on the 'my_feature' branch:

```
>> pip install -e .
```

This will install `drizzlepac` from this cloned source code in 'editable' mode, meaning that you can import the code from this directory when within a Python session. This makes it easier for development because you can have the code you're editing somewhere convenient in your file system vs. with other packages in 'site-packages'. If you cloned the repository on your Desktop, for example, you can modify it there and Python will know that is where the source code is when you're importing it within a Python session.

Note : If you use it, make sure to install iPython in your new environment as well. Otherwise, it will pick up packages from the base environment instead.

Step 4: Making code changes

Now that you've forked, cloned, made a new branch for your feature, and installed it in a new environment for development of `drizzlepac`, you are ready to make changes to the code. As you make changes, make sure to `git commit -m <"some message">` frequently (in case you need to undo something by reverting back to a previous commit - you cant do this if you commit everything at once!). After you've made your desired changes, and committed these changes, you will need to push them online to your 'remote' fork of `drizzlepac`:

```
>> git push origin my_feature
```

If the changes are significant, please make an entry in `CHANGELOG.rst` in the top level `drizzlepac` directory with a short description of the changes you've made and, once you open a pull request, add the corresponding PR number.

Step 4: Opening a pull request

Now, you can open a pull request on the main branch of the upstream `drizzlepac` repository.

1. On the `spacetelescope/drizzlepac` web page, after you push your changes you should see a large green banner appear at the top prompting you to open a pull request with your recently pushed changes. You can also open a pull request from the pull request tab on that page. Select your fork and your 'my_feature' branch, and open a pull request against the 'main' branch.
2. There is now a checklist of items that need to be done before your PR can be merged.
   - The continuous integration (CI) tests must complete and pass. The CI runs several different checks including running the unit tests, ensuring the documentation builds, checking for code style issues (see the PEP8 style guide), and ensuring any changes are covered by unit tests. The CI runs upon opening a PR, and will re-run any time you push commits to that branch.
   - You will need to add a change log entry in CHANGES.rst if your contribution is a new feature or bug fix. An entry is not required for small fixes like typos.
   - Your PR will need to be reviewed and approved by at least one maintainers. They may require changes from you before your code can be merged, in which case you will need to go back and make these changes and push them (they will automatically appear in the PR when they're pushed to origin/my_feature).

1.3.2 Advanced Contribution Instructions

Keeping your development branch current - rebasing

As `drizzlepac` is constantly evolving, you will often encounter the situation where you've made changes to your branch off 'main', but in the time its taken you to make those changes, 'upstream/main' has evolved with new commits from other developers. In this situation, you will want to make sure you incorporate these changes into your branch. Rebasing allows you to do two things - 1. apply others changes on top of yours, and 2. squash your commits, even if there aren't new changes to apply.

Periodically, while writing code, to keep your branch up to date you will want to do an interactive rebase against upstream/main to apply any new changes on top of yours:

```
>> git rebase -i upstream/main
```

This will then prompt you to select commits and commit messages - if you select just the top commit, this will 'squash' the others and combine them into one. You will be prompted to resolve any conflicts if you've made modifications to the same file someone else has. Once you've completed your rebase, you must `force push` your branch to origin/my_feature to make your local and remote match.

```
>> git push -f origin/my_feature
```

Before merging a PR, we typically would like you to rebase and squash all of your commits into a single one, which is also done with `git rebase`

Writing and building documentation

`drizzlepac` uses sphinx to generate documentation, which is then hosted online on readthedocs.

You can access the documentation on the DrizzlePac readthedocs website - the 'latest' version is whatever is currently on the main branch. If you successfully merge a PR with documentation changes, they will only appear on 'latest' until the next DrizzlePac release.

All of the documentation resides in the `drizzlepac/docs` subdirectories (mainly within directories in `drizzlepac/docs/drizzlepac`, organized by module). The documentation is written in `.rst` (reStructured text) files - if you wish to make changes or add to the documentation, do so in these files in the appropriate location. reStructured text is the markup language used by sphinx, for information on the syntax refer to the sphinx documentation.

While writing documentation, you will want to make sure the documentation builds successfully (i.e, produces a working .html file). This happens on the CI when you open a pull request, but it is a good idea to build the documentation yourself locally as well. Before you do this, you will need to make sure you have the correct dependencies installed in your current environment. All of these optional dependencies are specified in `pyproject.toml` and include things like the correct version of sphinx, as well as the necessary sphinx themes that the project uses. These do not install automatically when you install `drizzlepac` unless directly specified. To do this, while in the top level directory of `drizzlepac` on your my_feature branch:

```
>> pip install -e ".[docs]"
```

Now, with the correct documentation dependencies installed, you can attempt to build the documentation locally. To do this, enter into the `drizzlepac/docs` subdirectory and do:

```
>> make html
```

If the documentation successfully builds, the output HTML files will be output in the `_build/html` subdirectory. You can open the main `index.html` file in your browser and explore the full documentation pages just like the readthedocs site. If there were any errors or warnings, a traceback will be printed on your terminal. Small errors, like a typo in a link to another section, can cause frustrating errors so it is good practice to build the docs frequently when editing them.

Writing and running unit tests

Unit tests are located in the `tests` directory and are separated by instrument, with additional separate directories for Hubble Advanced Products (HAP) and the drizzle algorithm. These tests run the code on simplified datasets to make sure there are no breaking changes introduced. We aim to cover most lines of `drizzlepac` with unit test, so when adding code you will often need to write a new test or add to an existing test to ensure adequate coverage.

Take a look around at the existing tests - many tests use a @pytest.fixture to set up a common dataset. The test functions themselves set up a scenario to run a function under certain conditions, and culminate in a set of assertions that need to pass (or fail if the test is marked as `xfail`).

The CI will run the unit tests on your branch when you open a pull request. They will re-run every time you push commits to this remote branch as well. Unit tests must pass on any changes you make, and if you're introducing new code that isn't covered by an existing test, you will need to write one. The `codecoverage`

check that runs on the CI when you open a pull request will tell you if you've introduced any new lines that aren't covered by a test.

You can also run unit tests locally, and you should do this periodically to test your code. To do this, you will need to install the optional dependencies needed for running tests.

```
>> pip install -e ".[test]"
```

This will install the optional 'test' dependencies specified in `pyproject.toml` that don't install by default. The package `pytest` is one of these and is what's used to run the tests. `pytest` searches through all the directories in your repository (underneath the directory from which it was invoked command line) and looks for any directories called 'test' or .py files with the word 'test' in the name. Functions in these files will be executed.

To run all of the `drizzlepac` unit tests, while in the `drizzlepac/` level directory, simply run the command:

```
>> pytest
```

If you want to run all the tests for a single instrument/directory, for example `WFC3`, you can run this from 'drizzlepac/tests/wfc3'.

To run all tests within a single test file (for example, all tests in `test_wfc3`).

```
>> pytest drizzlepac/tests/wfc3/test_wfc3.py
```

Configuring pytest for unit tests

`pytest` can be configured with many different flags - see the pytest documentation to see all of these. Here we will summarize a few of the most useful options.

If you are writing and debugging a test, you may find it helpful to have print statements printed to the terminal while running tests (which doesn't happen by default). To do this,

```
>> pytest -s
```

If you want to only run a specific test function within a file (or only tests with names that contain a certain string):

```
>> pytest -k testname.
```

This will search all files under the directory you're in for files or functions with the string 'test' in them, and within those files run only the test functions that contain the string `testname`.

Within the test files themselves, decorators can be used to control the behavior of the test. Some of the more useful decorators include: 1. @pytest.mark.parametrize can be used to run a single test on multiples sets of input parameters 2. @pytest.skip can be used to skip tests altogether, or under specific conditions (for example, only when being run by the CI) 3. @pytest.fixture to declare a fixture 1. @pytest.mark.xfail will make a test pass only if it fails.

STScI Regression Tests

In addition to unit tests, `drizzlepac` has a set of regression tests. Many of tests must be run on the STScI system and are not available to the public. These internal tests can be run using the following command, and as the names suggest, require storage space (~8 GB) and time (~3-4 hours) to run.

```
>> pytest --bigdata --slow
```

Running the pytests without these options will result in a number of tests being skipped and labeled as expected failures (xfails).

Simultaneously developing `drizzlepac` and one of its dependencies

If you encounter the scenario where you wish to simultaneously make changes in `drizzlepac` and also in one of its dependencies like `fitsblender` or `stsci.tools`, we recommend that you create a new environment with development versions of both of these packages. To do this, you can follow the same workflow outlined in the 'Contributing code' section of this guide. To summarize, you will want to create a new Python environment (called, for example, drizzlepac_fitsblender_dev), fork and clone a local copy of `drizzlepac` if you haven't already and install this (doing `pip install -e .` in the top level directory of `drizzlepac`), and then fork and clone `fitsblender` and install it in this environment in the same way. To double check that you have the correct dev versions of these packages in your environment, you can check their versions by doing:

```
>> conda list
```

When opening up two dependent pull requests in `drizzlepac` and one of its dependency packages, unit tests will not pass on the CI because the `pyproject.toml` file in `drizzlepac` points to the last released version of `fitsblender`, and fitsblender points to the last version of `drizzlepac`, so the issue becomes circular. What you will need to do is modify the `pyproject.toml` files in both packages to point to the other to demonstrate that CI tests pass (and make a comment noting this in your PR), and then change it back before the PR is merge so that changes to `pyproject.toml` are not merged into main. In your `drizzlepac` branch, to point to your branch in the dependent package (in this example `fitsblender`), change the required `fitsblender` version in `pyproject.toml` to:

```
>> fitsblender @  git+https://github.com/<your_username>/fitsblender.git@<your_
↪branch>
```

And similarly, in `fitsblender`, change the required `drizzlepac` version to:

```
>> drizzlepac @  git+https://github.com/<your_username>/drizzlepac.git@<your_
↪branch>
```

Let the CI run and ensure it passes, comment this in your PR and make sure the reviewers confirm, and then change the versions back before your PR is merged (which will again cause the CI to fail, but that's OK).

1.3.3 Code of Conduct

We expect all "spacetelescope" organization projects to adopt a code of conduct that ensures a productive, respectful environment for all open source contributors and participants. We are committed to providing a strong and enforced code of conduct and expect everyone in our community to follow these guidelines when interacting with others in all forums. Our goal is to keep ours a positive, inclusive, successful, and growing community. The community of participants in open source Astronomy projects is made up of members from around the globe with a diverse set of skills, personalities, and experiences. It is through these differences that our community experiences success and continued growth.

As members of the community,

- We pledge to treat all people with respect and provide a harassment- and bullying-free environment, regardless of sex, sexual orientation and/or gender identity, disability, physical appearance, body size, race, nationality, ethnicity, and religion. In particular, sexual language and imagery, sexist, racist, or otherwise exclusionary jokes are not appropriate.
- We pledge to respect the work of others by recognizing acknowledgment/citation requests of original authors. As authors, we pledge to be explicit about how we want our own work to be cited or acknowledged.
- We pledge to welcome those interested in joining the community, and realize that including people with a variety of opinions and backgrounds will only serve to enrich our community. In particular, discussions relating to pros/cons of various technologies, programming languages, and so on are welcome, but these should be done with respect, taking proactive measure to ensure that all participants are heard and feel confident that they can freely express their opinions.
- We pledge to welcome questions and answer them respectfully, paying particular attention to those new to the community. We pledge to provide respectful criticisms and feedback in forums, especially in discussion threads resulting from code contributions.
- We pledge to be conscientious of the perceptions of the wider community and to respond to criticism respectfully. We will strive to model behaviors that encourage productive debate and disagreement, both within our community and where we are criticized. We will treat those outside our community with the same respect as people within our community.
- We pledge to help the entire community follow the code of conduct, and to not remain silent when we see violations of the code of conduct. We will take action when members of our community violate this code such as such as contacting conduct@stsci.edu (all emails sent to this address will be treated with the strictest confidence) or talking privately with the person.

This code of conduct applies to all community situations online and offline, including mailing lists, forums, social media, conferences, meetings, associated social events, and one-to-one interactions.

Parts of this code of conduct have been adapted from the Astropy and Numfocus codes of conduct. http://www.astropy.org/code_of_conduct.html https://www.numfocus.org/about/code-of-conduct/

1.3.4 License

```
BSD 3-Clause License

Copyright (c) 2024, Association of Universities for Research in Astronomy (AURA)
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this
  list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice,
  this list of conditions and the following disclaimer in the documentation
  and/or other materials provided with the distribution.

* Neither the name of the copyright holder nor the names of its
  contributors may be used to endorse or promote products derived from
  this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

1.3.5 Requirements

```
    'astropy>=5.0.4',
    'fitsblender>=0.4.2',
    'scipy',
    'matplotlib',
    'stsci.tools>=4.0',
    'stsci.image>=2.3.4',
    'stsci.imagestats',
    'stsci.skypac>=1.0.9',
    'stsci.stimage',
    'stwcs>=1.5.3',
```

(continues on next page)

(continued from previous page)

```
    'tweakwcs>=0.8.2',
    'stregion>=1.1.7',
    'requests',
    'scikit-learn>=0.20',
    'bokeh',
    'pandas',
    'spherical_geometry>=1.2.22',
    'astroquery>=0.4',
    'astrocut<=0.9',
    'photutils>1.5.0',
    'lxml',
    'PyPDF2',
    'scikit-image>=0.14.2',
    'tables',
    'typing_extensions>4.0',
    'markupsafe<=2.0.1',
```

1.3.6 DrizzlePac Release Notes

The version of DrizzlePac can be identified using

```
> python
>>> import drizzlepac
>>> drizzlepac.__version__
```

The following notes provide some details on what has been revised for each version in reverse chronological order (most recent version at the top of the list). The number at the end of each item is the Github Pull Request (PR) number of the code change for that issue. These PRs can be viewed at:

https://github.com/spacetelescope/drizzlepac/pulls

3.7.1 (unreleased)

- Added contributors guide to readthedocs. [#1787]
- Removed "tophat" as a kernel option, added warnings for "gaussian" and "lanczos3" that they may not be conserving flux. [#1786]
- Updated config json to exclude bad pixels in single WFC3/IR SVM processing. [#1783]
- Bug fix for mdriztab=True option in Astrodrizzle previously overwriting user inputs. [#1774]
- Reverted PR #1222 allowing pixels to be filled with available data where WHT=0. [#1767]
- Force the identified bad rows to be removed from the total (aka white light) source catalog before the corresponding bad segments are removed from the segmentation image. [#1771]
- Improved calculation of S_REGION using dialation and erosion. [#1762]

- Skycell added to flt(c) and drz(c) science headers for the pipeline and svm products. [#1729]

3.7.0 (02-Apr-2024)

- Update project.toml file to specify numpy>=1.18, <2.0 [#1743]
- Update project.toml file to specify python_requires>=3.10 [#1737]
- Github branch "master" renamed to main. [#1725]
- Clean up spacing in toml file to eliminate improper spacing to avoid decprecation warning [#1731]
- Clean up YAML diagram in of workflows area [#1728]
- Updated installation instructions and small text changes [#1727]
- Remove outdated references of Pyraf and change to Python [#1726]
- Fix to add "stregion" to the requirements-dev.txt file to fix the build error under Python 3.12. [#1714]
- Reorganized the readthedocs documentation with the help of various STScI staff. [#1717]
- Updates requirements-dev.txt to not install eggs that cause problems for the regression tests [#1721]
- Regression Testing: allow "dev" jobs to fail [#1718]
- Initial setup for Architectural Design Records used to keep track of top-level thinking behind the code. [#1697]

3.6.2 (27-Nov-2023)

- At this time pin Astrocut to versions <=0.9 to avoid conflicts with urllib3 package. [#1689]
- Added functionality to allow the use of a two-column poller file. This is used to update the WFPC2 SVM aperture header keywords from the values in the poller file. [#1683]
- Removed the version restriction on matplotlib. [#1649]
- Forced a preferential order on the final selection of the WCS solution from the common pool of solutions among all input exposurea. All input images need to have the same WCSNAME (same WCS solution) when performing pipeline alignment to avoid imprinting differences from one catalog to another on the final fit and destroying the relative alignment. [#1645, #1638]
- Redesigned the overall structure of the documentation, readthedocs, for the package. [#1620]
- Addressed a bug in the calculation of measurements for each detected source in the filter catalogs. The detection catalog, based upon the "total" image, is now used in the correct manner to define the source centroids and shape properties. In addition, these properties are used to perform aperture photometry. [#1614]
- Updated the HAP drizzle parameters for WFPC2. The primary change includes changing skymethod='localmin' from the prior 'match' which did not work well for the overlapping chips. [#1617]

- Corrected reference catalog weights from being proportional to sigma to the proper 1/sigma**2. [#1616]
- Removed the use of the shadow mask as an initial step in addressing the WFPC2 chip gaps [#1551]
- Fixed a bug in processing of the `group` argument due to which the code would crash when `group` would be an integer number or a list of numbers. Also, added support for specifying extensions as tuples of `(extname, extver)`. [#1612]

3.6.1 (15-Jun-2023)

- Fixed an incompatiblity in the `minmed` code for cosmic ray rejection with the `numpy` version >=`1.25`. [#1573]
- Fixed projection cell identification in overlapping regions. [#1572]
- Force the version of matplotlib to be <= 3.6.3 as the newer versions of the library cause problems with the calcloud preview generation. [#1571]

3.6.0 (12-Jun-2023)

- Modified the pyproject.toml file to ensure the tweakwcs version is greater than 0.8.2 as the issue of taking a very long time to compute the bounding polygon now defaults to an approximate method which is significantly faster. [#1565]
- Modified Projection Cell 0 declination coordinate of the center to be -89.999999999997 and the Projection Cell 2643 declination coordinate to be 89.999999999997 to shift the WCS CRVAL position slightly off the pole. [#1560]
- Modified the criteria for the rejection of catalogs based upon the cosmic ray criterion. An empty catalog (n_sources=0) should not be rejected by the CR contamination. Also, if a catalog is empty, it should not trigger the rejection of the other "type" of catalog (type=point vs segment). [#1559]
- For WFPC2 datasets which turn out to have no viable data to process and a manifest file has been requested, force an empty manifest file to be generated and issue the exit code NO_VIABLE_DATA (65). [#1550]
- Protect against writing the S_REGION keyword in intentionally empty DRZ/DRC files in `processinput.process` to avoid messy crash. [#1547]
- Fix a bug in `processinput.buildFileListOrig` due to which astrodrizzle might crash when `updatewcs` is set to `True`. [#1549]
- Turn off use of `verify_guiding()` for WFPC2 images only as its use incorrectly recognizes diffraction spikes from saturated stars as evidence of loss of lock and flags those exposures as 'bad'. [#1511]
- Ensure processing of all IMAGETYP=EXT WFPC2 targets. [#1505]
- Properly identify neighbor Projection Cells which overlap input exposures. [#1503]
- Updates identify and remove any WFPC2 calibration exposures that cannot be processed during standard pipeline alignment and drizzling. The list of recognized calibration target names was updated to

accommodate WFPC2 and to identify exposures to be skipped and deleted after converting the D0M images into FLT images. [#1514]

- Compute a default kernel for use with astrometric_utils.extract_sources() function when the kernel parameter is None. The default kernel is based on the fwhm parameter of the same function. [#1519]
- Address many ReadTheDocs issues. [#1521 - #1529]
- Write the EXPNAME keyword to the ACS SVM and MVM headers to avoid errors and enforce consistency with WFC3. [#1530]
- Properly populate the S_REGION keyword with a closed polygon for the pipeline FLT/FLC images. [#1533]
- Compute the S_REGION values for pipeline drizzled products. [#1535]
- Ensure the DATE keyword is written to the primary header of all output drizzled products. The DATE represents the date the file was written. [#1537]
- Update to ensure the SVM FLT/FLC files all contain the S_REGION keyword and the value of the keyword is a closed polygon. [#1536]

3.5.1 (08-Feb-2023)

- Turn on use of `verify_guiding()` to ignore exposures where guide star lock was lost and the stars are trailed. [#1443]
- Ensure when no sources are found and the variable thresh is zero, the `verify_crthesh()` properly indicates the catalog failed the CR threshold. [#1450]
- Added informational text when the catalog service fails (e.g., service cannot be reached or the request was somehow malformed) to make the default response more helpful. The request specification is also sent to the log, so the user can see what was actually requested. [#1451]
- Protect against there being no sources left to measure the properties after cleaning cosmic rays from the input in `verify_guiding()`. [#1466]
- Check the SCI extension(s) of the output FLT/FLC and DRZ/DRC files. If the active WCS solution is 'a priori', delete the following keywords if they are associated with the active WCS as they are residue from a previous 'a posteriori' solution: NMATCHES, RMS_RA/RMS_DEC, FITGEOM, and CRDER1/CRDER2. Ensure the WCSTYPE is based upon the active WCSNAME to clean up any confusion. [#1465]
- Protect against inability to find a FWHM due to a fitting problem. [#1467]
- Implement photometric equalization for standard pipeline processing (runastrodriz) of WFPC2 data. [#1471]
- Update required to the compute_2d_background() function of the astrometric_utils module to accommodate changes in the PhotUtils API. [#1480]

3.5.0 (10-Oct-2022)

- Introduced a new `apply_tweak()` function as a replacement to the `tweakback()`. `apply_tweak()` preserves the functionality of `tweakback` with a re-designed API. Existing `tweakback` was deprecated. [#1372]
- Updated segmentation source catalog generation to use ICRS as input RADESYS when input images have an unsupported REFFRAME value (like OTHER or B1950). [#1423]
- Refactored code to work with changes in `tweakwcs` version 0.8.0. [#1430]
- Ignore non-CTE-corrected exposures when SVM or MVM products also include CTE-corrected exposures as inputs. [#1433]

3.4.3 (24-Aug-2022)

This release includes includes updates for these features in addition to various bug fixes:

- Initial support for aligning and creating SVM and MVM products for WFPC2 data based on unoptimized processing parameters
- Python 3.10 support
- Photutils 1.4.0 (and newer) support
- Updated documentation on SVM processing and output mosaics

The list of specific changes for the significant issues includes:

- Fixed skycell size in pixels as quoted in the documentation. (#1387)
- Ensure Ramp filter data is not used for MVM processing (#1393)
- Added requested values and clarification text regarding photometry to the catalogs (#1390)
- Modified the docstring which defines the HAPLEVEL and its associated meaning (#1395)
- Modified the "exposure level" products to have a HAPLEVEL = 1 (#1398)
- Get full S_REGION outline (#1401)
- Update readthedocs for SVM catalog generation (#1400)
- Delete all reference catalogs during SVM processing (#1409)
- Update runastrodriz to work with WFPC2 data as singletons (#1412)
- Revert sky matching to use local sky minimization upon any error (#1411)
- Update SVM to support processing WFPC2 exposures (#1418)
- Add support for Python 3.10 (#1420)
- Add WFPC2 support for MVM processing (#1422)
- Support additional RADESYS options for input files (#1423)
- Ensure the gain variables are defined for all detectors (#1425)

- Essentially remove restriction on PhotUtils package version (#1426)

3.4.2 (27-May-2022)

This release addresses a number of issues related to SVM and MVM processing.

- Reset tasknames to work with TEAL (#1285)
- Protect computations when photflam is equal to 0.0 (#1295)
- MVM: Define MVM-specific processing parameters for drizzling (#1277)
- Remove IPPPSSOO keyword from MVM product headers (again) (#1297)
- Fix problem with astropy 5.0 table interpretation (#1292)
- Statistics for SVM and MVM (#1300)
- SVM: add/remove/update Astrodrizzle Parameter files (#1303)
- Explicitly update boolean column in ASN tables (#1307)
- Synchronize output WCS specifications for SVM processing (#1312)
- Smooth out determination of S_REGION vertices (#1315)
- Ensure units of catalog variables comply with Astropy (#1316)
- Apply default alignment fit parameters for zero exptime exposures (#1319)
- Fix bug caused by Astropy Tables being interpreted as QTables (#1320)
- Revise logic for when mask keywords are computed (#1323)
- Restrict version of Photutils to < 1.4.0. (#1326)
- Add MEANWHT and MEDWHT keywords to drizzle products (#1324, #1349)
- Add documentation describing mvm products and artifacts (#1322)
- Add release notes for 3.4.1final (#1328)
- Fix typo in ACS MVM header rules file (#1332)
- Update astropy min version to 5.0.4 (#1335)
- Avoid archiving duplicate WCS solutions in SVM processing (#1333)
- Update installation dependencies for fitsblender and skypac (#1354)
- Flag and ignore bad images based on detecting linear features (#1351)
- Improve algorithm for identifying and filtering large segments (#1357)
- Carry over IDCSCALE keyword when updating WCS to match Grism WCS (#1355)
- Ignore MVM layers with no overlapping exposures (#1360)
- Update crder units (#1362)

- This change addresses bugs associated with the big_segments attribute of the segmentation image (#1365)
- Update the WFC3 rules files (#1366)
- Only allow "verify_guiding" check for MVM processing (#1368)
- Fix the size of the HAPEXPNAME column in the HDRTAB of the MVM output DRZ/DRZ file (#1371)
- Pass along default WCSNAME (#1370)
- Re-design tweakback (#1372)
- Bugfix: point-cat-fxm files being left around (#1369)

3.4.1 (5-Apr-2022)

This release addresses issues found in v3.4.0. The most significant issues were:

- Add documentation describing mvm products and artifacts (#1322)
- Revise logic for when mask keywords are computed (#1323)
- Restrict version of Photutils to < 1.4.0. (#1326)
- Add MEANWHT and MEDWHT keywords to drizzle products (#1324)
- Modify the units of the catalog variables so they are astropy-compatible (#1318)
- Smooth out determination of S_REGION vertices (#1315)
- Apply default alignment fit parameters for zero exptime exposures (#1319)
- fix for tasknames to once again work with TEAL (#1289)
- Revise code to properly support Astropy v5.0 (#1286 , #1290 , #1292, #1296, #1307)
- Protect computations in catalog generation when photflam is equal to 0.0 (#1295)
- Define MVM-specific and SVM-specific processing parameters for drizzling (#1277, #1303)
- Remove IPPPSSOO keyword from header of output SVM or MVM drizzle products (#1297)
- Insure correct statistics are reported in MVM headers (#1300)

3.4.0 (7-Mar-2022)

This major release adds support for multi-visit mosaic (MVM) processing, in addition to including numerous revisions to try to align more datasets successfully to GAIA during pipeline and single-visit mosaic (SVM) processing. Multi-visit mosaics (MVM) introduce the concept of SkyCells with new code added to define them. SkyCells are subarrays of pre-defined tangent planes spaced regularly on the sky as standardized definitions of mosaics to be created from all HST observations taken of each part of the sky.

New features added in this version include:

- Support for creating MVMs as generated by the 'drizzlepac/hapmultisequencer.py' module or using the new command-line task `runmultihap`.
- Tools for generating cutouts of MVM products found in the `drizzlepac/haputils/hapcut_utils.py` module.

The most significant revisions and bug fixes that affect output products of this version of the code include:

- Detect extension name from WFPC2 flat-field files. [#1193]
- Refactored the build system to be PEP-517 ad PEP-518 compliant. [#1244]
- Fixed a bug in the drizzle algorithm due to which input pixels with zero weights may still contribute to the output image. [#1222]
- Added Sphinx documentation describing tools used for working with MVM products. [#1144, #1150]
- Changed names of "ISO" columns in Segmentation catalog to be unique [#1155]
- Add WCS keyword values to catalog metadata [#1160]
- Enforced a minimum number of cross-matches for alignment to be 4 sources [#1187, #1218]
- Revised 2D background determination for smaller detectors to improve source detection during alignment. [#1187]
- Create empty catalogs when exposures are effectively blank. [#1199]
- Cut processing time from days to minutes for exposures of crowded fields of faint sources or fields dominated by a single large extended source. [#1198]
- Report correct value of NMATCHES keyword for number of sources actually used in alignment fit to GAIA. [#1217]
- Prevent older distortion models from overriding new distortion models when performing a posteriori alignment to GAIA. [#1220]
- Add explicit dependency on spherical-geometry package. [#1232]
- Update how make_poller_files.py generates visit numbers. [#1221]
- Insure both FLT and FLC headers have same a posteriori fit keywords. [#1238]
- MVM: Make tool to quantify quality of GAIA alignment generic for general use. [#1241]
- Fix logic to not align grism data in standard pipeline. [#1243]
- Remove nictools as a dependency for this package. [#1245]
- RickerWavelet Kernel for SBC to separate crowded PSFS needs to have dimensions which are odd [#1246]
- Refine headers for filter and total products to allow keywords like IPPPSSOO and ASN_ID which only apply to single exposures (or data from the same ASN) to be removed from SVM filter and total drizzle products and from MVM layers drizzle products [#1249]
- Remove logic from align that related to checking for alignment results in align.py when it was not necessary so that more data can successfully align to GAIA. [#1250]
- Add support for using astropy 5.0. [#1280]

3.3.1 (19-Nov-2021)

This version provides bug fixes primarily for the single-visit mosaic (SVM) processing.

- Insure a compatible version of photutils gets installed. [#1151]
- Improve handling of segmentation catalog generation for mostly or completely blank images. [#1152]
- Changed default floating point value in catalogs from -9999.9 to -9999.0. [#1165]
- Avoid creating an empty manifest file when no images get drizzled by SVM processing, unless the visit was comprised solely of Grism/Prism data. [#1174, #1181]
- Update total catalog to only remove sources which were not measured successfully in any filter. [#1175]
- Fix the units of a few variables in the output Point and Segmentation catalogs [#1178]

3.3.0 (28-Sep-2021)

This version includes all the functionality needed to generate source catalogs, both point source and extended (segment) source catalogs, during single-visit mosaic (SVM) processing. In fact,

- Updated code to work with Python >= 3.7
- **GAIAeDR3** catalog now the initial catalog of choice for a posteriori alignment during standard pipeline processing, as well as for SVM/MVM processing.
- SVM/MVM processing will loop over catalogs, fit methods and fit geometries in looking for a successful fit, using the first successful fit it computes.
  - CATALOGS used: **GAIAeDR3**, **GSC242**, **2MASS** (in this order)
  - methods: relative, image-by-image
  - geometries: **rscale**, **rshift**, **shift** (each with different minimum cross-matches)
- SVM processing will always generate both point source and extended source catalogs, even if the catalogs contain no rows of sources and measurements.
  - point source catalog will be generated using TinyTim PSF-based detection
  - extended source (segment) catalog will only have sources larger than the PSF kernel deblended.
  - catalog columns will closely resemble the Hubble Legacy Archive (HLA) catalogs columns
- Grism/Prism exposures do not get aligned, but instead get the WCS correction from direct images
- Added logic to handle visits where there are only Grism/Prism exposures with no direct images
- `S_REGION` keyword:
  - added to FLT/FLC file headers
  - revised region computation to match closely the actual exposure footprint within mosaic
- Always runs `updatewcs` on input files to insure pipeline-default WCSs are always present
  - Add `WCSNAME=OPUS` if no `IDCTAB` WCS was created by `updatewcs` (`NGOODPIX=0`, ...).

These changes, and additional significant bug fixes, were implemented using the following github PRs:

- Implemented deblending of segmentation source catalogs ONLY for sources larger than the PSF kernel. [#1131]
- Insure SVM processing always generates point-source and segmentation (extended) source catalogs, even if empty. [#1129]
- Implemented an efficient single-image identifier of possible cosmic-rays/defects, and applied it to help make image alignment more reliable. [#1129]
- Update logic for fitting between source lists to minimize/eliminate use of fitting with less than 4 sources. [#1129]
- Implemented model PSF-based point-source identification for SVM point-source catalog generation. [#903, #971, #1127]
- Removed dependence on private photutils functions while enabling support for all photutils versions >= 1.0.0. [#1127, #1117, #1116, #1096]
- Set values for crowding, biggest source, and source fraction for use when to use the RickerWavelet kernel and when to deblend sources when identifying extended sources using segmentation for the segment catalog. [#1115]
- Implemented a more efficient algorithm based on Harris corner detection for computing the `S_REGION` keyword for pipeline and SVM drizzle products. [#1106]
- Fix a memory corruption issue in `interpolate_bilinear()` in `cdrizzleblot.c` which could result in segfault. [#1048]
- Fixed multiprocessing incompatibility with `Python >= 3.8`. [#1101]
- Add support for environment variable switch, `PIPELINE_RESET_IDCTAB`, to `runastrodriz` which will automatically reset `IDCTAB` in FLT/FLC files if different from `IDCTAB` in RAW files. [#1046]
- Update documentation based on revisions to the code. [#941, #947, #953]
- Update default astrometry catalogs for alignment to try alignment to the `GAIA eDR3` catalog first. [#986, #1012]
- Enable user epoch selection when a user requests a GAIA catalog from the astrometry catalog web service. [#1006]
- Insure that `HDRNAME` is always valid for updated WCS solutions. [#966]
- Revised `S_REGION` keyword value to reflect actual outline of chips in drizzle products. [#951]
- Sky Subtraction step will automatically downgrade from `match` to `localmin`, and from `globalmin+match` to `globalmin` when sky matching runs into an Exception. [# 1007]
- Changed to insure that `EXTNAME` and `EXTVER` are always removed from simple FITS drizzle product headers. [#954]
- Changed to insure that all the distortion keywords (e.g., `TDD*`, `D2IM*`,...) are removed from from the output drizzle product headers [#954].
- Set a common active WCS for direct as well as corresponding Grism/Prism images [#929, #946]

- Fix a bug in `tweakback` that may cause incorrect "updated" WCS to be picked up from the drizzled image. [#913]
- Added `DRIZPARS` keyword to final output drizzle product primary header to document the name of the associated trailer file. [#934, #1078]

In addition, numerous changes were made to insure this code stayed compatible with numpy versions > 1.20 and astropy versions > 4.1.

Updates to the `STWCS` package version >= 1.6.0 also translated to the following changes to the Drizzlepac processing: - Insure HDRNAME keyword is never empty - Remove duplicate headerlet extensions when running updatewcs - Compute new a priori WCS solutions for new IDCTAB not already in astrometry database

API Changes:

imageObject.py:

- **class imageObject**: Added parameter `output` to enable determination of rootname for use in processing of each detector.

adrizzle.py:

- **drizSeparate**: Added optional parameter `logfile` for specifying what file to use for log messages.
- **drizFinal**: Added optional parameter `logfile` for specifying what file to use for log messages.

wcs_functions.py:

- Removed `hdulist` as parameter from `get_hstwcs`.

haputils/analyze.py:

- **analyze_data**: Added parameter `type` to customize logic for SVM processing.

haputils/astrometric_utils.py:

- **retrieve_observation**: Added parameter `product_type` to allow for selection of type of products to be returned; pipeline, HAP, or both.

haputils/make_poller_files.py:

- New function `generate_poller_file` added to create inputs for SVM processing from files on disk.

haputils/processing_utils.py:

- New function `find_footprint` added to determine corners of all chips in an image for computation of `S_REGION` keyword.
- New function `interpret_sregion` added to convert `S_REGION` keyword value into list of RA/Dec points for visualization.

3.2.1 (16-Feb-2021)

- Fix problems with testing code for this package [#940]

3.2.0 (7-Dec-2020)

This version provides the first operational implementation of the single-visit mosaic processing used to create the single-visit mosaics products.

- revise naming convention for the StaticMask file so that it has a dataset-specific name instead of a generic common name. [#876]
- Update `runastrodriz` to work under Windows while adding documentation to tell the user to run with `num_cores` set to 1. [#794]
- Fixed a bug in `TweakReg` due to which `TweakReg` would crash when `updatehdr` was set to `False`. [#801]

3.1.8 (11-Aug-2020)

A number of changes have been implemented to either correct problems or improve the processed results. The most significant of the changes are:

- rscale only used for alignment.
- a minimum of 6 sources now gets used for alignment
- no proper motions used in astrometric (GAIA) catalog when attempting a posteriori fitting
- chip-to-chip alignment errors were corrected

In addition to a few dozen bug fixes, the following updates to the algorithms were also implemented.

- Simplified the logic in `tweakreg` for deciding how to archive primary WCS resulting in a reduction of duplicate WCSes in image headers. [#715]
- Added polynomial look-up table distortion keywords to the list of distortion keywords used by `outputimage.deleteDistortionKeywords` so that distortions can be removed from ACS images that use `NPOLFILE`. This now allows removal of alternate WCS from blotted image headers. [#709]
- Added `rules_file` parameter to AstroDrizzle to enable use of custom files in pipeline processing. [#674]
- Only apply solutions from the astrometry database which were non-aposteriori WCS solutions as the PRIMARY WCS. This allows the pipeline to compare the true apriori WCS solutions (e.g., GSC or HSC WCSs) to aposteriori solutions computed using the latest distortion-models and alignment algorithms being used at the time of processing. [#669]
- Verification using a similarity index gets reported in the trailer file and does not get used as a Pass/Fail criteria for alignment. [#619]
- If verification fails for either pipeline-default or apriori solution, reset cosmic-ray(CR) flag (4096) in DQ arrays. This will allow subsequent attempt to align the images to not be impacted by potentially

mis-identified CRs that most likely blanked out real sources in the field. As a result, the image alignment process became more robust when computing the aposteriori alignment. [#614]

- Fix a crash in `tweakreg` when finding sources in very large images due to a bug in `scipy.signal.convolve2d`. [#670]
- Fix a bug in `tweakreg` due to which the number of matched sources needed to be *strictly* greater than `minobj`. Now the minimum number of matched sources maust be *at least* equal or greater than `minobj`. [#604]
- Fix a crash in `tweakreg` when `2dhist` is enabled and `numpy` version is `1.18.1` and later. [#583, #587]
- Update calibrated (FLC/FLT) files with RMS and NMATCH keywords when it successfully aligns the data to GAIA using the a posteriori fit. Headerlet files for this fit which already have these keywords are now retained and provided as the final output headerlets as well. [#555]
- Insure HDRNAME keyword gets added to successfully aligned FLC/FLT files. [#580]
- Fix problem with 'tweakback' task when trying to work with updated WCS names. [#551]
- Fix problems found in processing data with NGOODPIX==0, DRC files not getting generated for singletons, alignment trying to use a source too near the chip edge, catch the case were all inputs have zero exposure time, lazily remove alignment sub-directories, fixed a bug in overlap computation that showed up in oblong mosaics, recast an input to histogram2d as int, defined default values for tables when no sources were found. [#593]
- Updated to be compatible with tweakwcs v0.6.0 to correct chip-to-chip alignment issues in aposteriori WCS solutions. [#596]
- Correctly define output drizzle product filename during pipeline processing for exposures with 'drz' in the rootname. [#523]
- Implement multiple levels of verification for the drizzle products generated during pipeline processing (using runastrodriz); including overlapp difference computations [#520], and magnitude correlation [#512].
- Replace alignimages module with O-O based align [#512]
- Fix problem with NaNs when looking for sources to use for aligning images [#512]
- Fixed code that selected the brightest sources to use for alignment allowing alignment to work (more often) for images with saturated sources. [#512]
- Use logic for defining the PSF extracted from the images to shrink it in each axis by one-half for images of crowded fields to allow for more sources to be extracted by daofind-like algorithm. This enables source finding and alignment to work more reliably on crowded field images. [#512]
- Insure all input files, especially those with zero exposure time or grism images, get updated with the latest pipeline calibration for the distortion. [#495]

This version also relies on updates in the following packages to get correctly aligned and combined images with correctly specified WCS keywords:

- TWEAKWCS 0.6.4: This version corrects problems with the chip-to-chip separation that arose when applying a single fit solution to the entire observation.

- STWCS 1.5.4: This version implements a couple of fixes to insure that use of headerlets defines the full correct set of keywords from the headerlet for the PRIMARY WCS in the science exposure without introducing multiple copies of some keywords.
- Numpy 1.18: Changes in numpy data type definitions affected some of the code used for computing the offset between images when performing aposteriori alignment during pipeline processing and when running the 'tweakreg' task.

3.1.3 (5-Dec-2019)

- Fixed a bug in the `updatehdr.update_from_shiftfile()` function that would crash while reading shift files. [#448]
- Migration of the HAP portion of the package to an object-oriented implemenation. [#427]
- Added support for providing HSTWCS object as input to 'final_refimage' or 'single_refimage' parameter. [#426]
- Implementation of grid definition interface to support returning SkyCell objects that overlap a mosaic footprint. [#425]
- Complete rewrite of `runastrodriz` for pipeline processing to include multi-level verification of alignment. [#440]

3.0.2 (15-Jul-2019)

- Removed deprecated parameter `coords` from the parameter list of `pixtopix.tran()` function. [#406]
- Modified the behavior of the `verbose` parameter in `pixtopix.tran()` to not print coordinates when not run as a script and when `output` is `None`. [#406]
- Fixed a compatibility issue in `tweakutils` that would result in crash in `skytopix` when converting coordinates in `hms` format. [#385]
- Fixed a bug in the `astrodrizzle.sky` module due to which sky matching fails with "Keyword 'MDRIZSKY' not found" error when some of the input images do not overlap at all with the other images. [#380]
- Fixed a bug in the `util.WithLogging` decorator due to which incorrect log file was reported when user-supplied log file name does not have `.log` extension. [#365]
- Fixed a bug introduced in #364 returning in `finally` block. [#365]
- Improved `util.WithLogging` decorator to handle functions that return values. [#364]
- Fixed a bug in the automatic computation of the IVM weights when IVM was not provided by the user. [#320]
- Fixed a bug in the 2D histogram code used for estimating shifts for catalog pre-matching. This may result in better matching. [#286]
- Now `tolerance` (in `tweakreg`) is no longer ignored when `use2dhist` is enabled. [#286]

- Fixed VS compiler errors with pointer artithmetic on void pointers. [#273]
- Fix logic so that code no longer tries to update headers when no valid fit could be determined. [#241]
- Fixed a bug in the computation of interpolated large scale flat field for STIS data. The bug was inconsequential in practice. Removed the dependency on `stsci.imagemanip` package. [#227]
- Removed the dependency on `stsci.ndimage` (using `scipy` routines instead). [#225]
- Added `'Advanced Pipeline Products'` alignment code to `drizzlepac` package. Enhance `runastrodriz` to compute and apply absolute astrometric corrections to GAIA (or related) frame to images where possible. [#200, #213, #216, #223, #234, #235, #244, #248, #249, #250, #251, #259, #260, #268, #271, #283, #294, #302]
- Add computation and reporting of the fit's Root-Mean-Square Error (RMSE) and Mean Absolute Error (MAE). [#210]
- Replaced the use of `WCS._naxis1` and `WCS._naxis2` with `WCS.pixel_shape` [#207]
- Removed support for Python 2. Only versions >= 3.5 are supported. [#207]
- Use a more numerically stable `numpy.linalg.inv` instead of own matrix inversion. [#205]
- The intermediate fit match catalog, with the name `_catalog_fit.match` generated by `tweakreg` now has correct RA and DEC values for the sources after applying the fit. [#200, #202]
- Simplify logic for determining the chip ID for each source. [#200]

2.2.6 (02-Nov-2018)

- Fix a bug that results in `tweakreg` crashing when no sources are found with user-specified source-finding parameters and when `tweakreg` then attempts to find sources using default parameters. [#181]
- Updated unit_tests to use original inputs, rather than updated inputs used by nightly regression tests.
- Fix `numpy` "floating" deprecation warnings. [#175]
- Fix incorrect units in CR-cleaned images created by `astrodrizzle`. Now CR-cleaned images should have the same units as input images. [#190]

2.2.5 (14-Aug-2018)

- Changed the color scheme of the `hist2d` plots to `viridis`. [#167]
- Refactored test suite
- `sdist` now packages C extension source code

2.2.4 (28-June-2018)

- Replace `pyregion` with `stregion`

2.2.3 (13-June-2018)

- Updated links in the documentation to point to latest `drizzlepac` website and online API documentation.
- Code cleanup.
- Updated C code to be more compatible with latest numpy releases in order to reduce numerous compile warnings.
- Updated documentation to eliminate (at this moment) all sphinx documentation generation warnings.
- Moved `'release_notes.rst'` to `'CHANGELOG.rst'` in the top-level directory.
- Improved setup to allow documentation build. See drizzlepac PR #142 and Issue #129 for more details.
- Fixed a bug in a print statement in the create median step due to which background values for input images used in this step were not printed.
- Fixed a bug due to which `TweakReg` may have effectively ignored `verbose` setting.
- Fixed a bug in `drizzlepac.util.WithLogging` due to which `astrodrizzle` would throw an error trying when to raise another error. See Issue #157 for more details.

2.2.2 (18-April-2018)

- Fixed a bug in `TweakReg` introduced in `v2.2.0` due to which, when `TweakReg` is run from the interpreter, the code may crash when trying to interpret input files.

2.2.1 (12-April-2018)

- Fixed problems with processing WFPC2 data provided by the archive. User will need to make sure they run `updatewcs` on all input WFPC2 data before combining them with `astrodrizzle`.

2.2.0 (11-April-2018)

- Implemented a major refactor of the project directory structure. Building no longer requires `d2to1` or `stsci.distutils`. Drizzlepac's release information (i.e. version, build date, etc) is now handled by `relic`. See https://github.com/spacetelescope/relic
- Added basic support for compiling Drizzlepac's C extensions under Windows.
- Documentation is now generated during the build process. This ensures the end-user always has access to documentation that applies to the version of `drizzlepac` being used.

- Swapped the effect of setting `configobj` to `None` or `'defaults'` in `AstroDrizzle` and `TweakReg`. When calling one of these tasks with `configobj` parameter set to `None`, values for the not-explicitly-specified parameters should be set to the default values for the task. When `configobj` is set to `'defaults'` not-explicitly-specified parameters will be loaded from the `~/.teal/astrodrizzle.cfg` or `~/.teal/tweakreg.cfg` files that store latest used settings (or from matching configuration files in the current directory). See https://github.com/spacetelescope/drizzlepac/pull/115 for more details.

2.1.22 (15-March-2018)

- Changed the definition of Megabyte used to describe the size of the buffer for create median step (`combine_bufsize`). Previously a mixed (base-2 and base-10) definition was used with 1MB = 1000x1024B = 1024000B. Now 1MB is defined in base-2 (MiB) as 1MB = 1024x1024B = 1048576B.
- Redesigned the logic in `createMedian` step used to split large `single_sci` images into smaller chunks: new logic is more straightforward and fixes errors in the old algorithm that resulted in crashes or unnecessarily small chunk sizes that slowed down `createMedian` step.
- Due to the above mentioned redesign in the logic for splitting large images into smaller chunks, now `overlap` can be set to 0 if so desired in the `minmed` combine type. Also, it is automatically ignored (set to 0) for all non-`minmed` combine types. This will result in additional speed-up in the Create Median step.
- Both `AstroDrizzle()` and `TweakReg()` now can be called with `configobj` parameter set to `'defaults'` in order to indicate that values for the not-explicitly-specified parameters should be set to the default values for the task instead of being loaded from the `~/.teal/astrodrizzle.cfg` or `~/.teal/tweakreg.cfg` files that store latest used settings.
- Updated documentation.

2.1.21 (12-January-2018)

- Restore recording of correct `EXPTIME` value in the headers of single drizzled ("single_sci") images. See https://github.com/spacetelescope/drizzlepac/issues/93 for more details.
- Fixed a bug in `drizzlepac` due to which user provided `combine_lthresh` or `combine_hthresh` in the `CREATE MEDIAN IMAGE` step were not converted correctly to electrons (processing unit). This bug affected processing of WFPC2, STIS, NICMOS, and WFC3 data. See https://github.com/spacetelescope/drizzlepac/issues/94 for more details.
- Modified print format so that scales, skew and rotations are printed with 10 significant digits while shifts are printed with 4 digits after the decimal point.

2.1.20 (07-October-2017)

- Fixed a bug in expanding reference catalog in `TweakReg` that would result in the code crashing. See https://github.com/spacetelescope/drizzlepac/pull/87 for more details.
- Fixed a bug due to which user catalog fluxes would be interpreted as magnitudes when `fluxunits` was set to `'cps'`. See https://github.com/spacetelescope/drizzlepac/pull/88 for more details.
- Fixed a bug due to which user-supplied flux limits were ignored for the reference catalog. See https://github.com/spacetelescope/drizzlepac/pull/89 for more details.

2.1.19 (29-September-2017)

- Fixed a bug in computing optimal order of expanding reference catalog that resulted in code crashes. See https://github.com/spacetelescope/drizzlepac/pull/86 for more details.

2.1.18 (05-September-2017)

- Fixed `astrodrizzle` lowers the case of the path of output images issue. See https://github.com/spacetelescope/drizzlepac/issues/79 for more details.
- Fixed `tweakreg` ignores user-specified units of image catalogs (provided through the `refcat` parameter) issue. See https://github.com/spacetelescope/drizzlepac/issues/81 for more details.
- Corrected a message printed by tweakreg about used WCS for alignment. Also improved documentation for the `refimage` parameter.

2.1.17 (13-June-2017)

- `drizzlepac.adrizzle` updated to work with numpy >=1.12 when they implemented more strict array conversion rules for math. Any input which still has INT format will be converted to a float before any operations are performed, explicitly implementing what was an automatic operation prior to numpy 1.12.

2.1.16 (05-June-2017)

- Fixed a bug introduced in release v2.1.15 in the logic for merging WCS due to which custom WCS scale was being ignored.

2.1.15 (26-May-2017)

- `fits.io` operations will no longer use memory mapping in order to reduce the number of file handles used when running either `astrodrizzle` or `tweakreg`. See issue #39 for more details.
- Fixed bugs and improved the logic for merging WCS that is used to define `astrodrizzle`'s output WCS.
- Added `crpix1` and `crpix2` parameters to custom WCS.

2.1.14 (28-Apr-2017)

- Supressed info messages related inconsistent WCS - see issue #60 and stwcs issue #25 for more details.

2.1.13 (11-Apr-2017)

- Fixed a bug due to which sky background was subtracted by `astrodrizzle` from the images even though `skysub` was set to `False` when `MDRIZSKY` was already present in input images' headers.

2.1.12 (04-Apr-2017)

- `astrodrizzle` now will run `updatewcs()` on newly created images when necessary, e.g., after converting WAVERED FITS to MEF format (`*c0f.fits` to `*_c0h.fits`) or after unpacking multi-imset STIS `_flt` files. See PR #56 for more details.
- Fixed a bug that was preventing processing STIS image data.
- Fixed a bug in reading user input (see issue #51).

2.1.11 (24-Mar-2017)

Bug fix release (a bug was introduced in v2.1.10).

2.1.10 (23-Mar-2017)

Some of the changes introduced in release v2.1.9 were not backward compatible. This release makes those changes backward compatible.

2.1.9 (22-Mar-2017)

Compatibility improvements with Python 3 and other STScI software packages.

2.1.8 (08-Feb-2017)

- Drizzlepac code will no longer attempt to delete "original" (WCS key 'O') resulting in a decreased number of warnings (see issue #35).
- Negative values are now zeroed in the 'minmed' step before attempting to estimate Poisson errors (see issue #22).
- Fixed a bug in `tweakreg` due to incorrect matrix inversion.
- Improved compatibility with `astropy.io.fits` ('clobber' parameter) and `numpy` which has reduced the number of deprecation warnings).
- Existing static masks in the working directory are now overwritten and not simply re-used (see issue #23).
- Corrected formula for σ computation in the "create median" step to convert background to electrons before computations. This bug was producing incorrect σ for instruments whose gain was different from one.
- Improved `astrodrizzle` documentation for `combine_type` parameter which now also documents the formula for σ computation when `combine_type` parameter is set to `'minmed'`.

2.1.6 and 2.1.7rc (15-Aug-2016)

Package maintenance release.

2.1.5 (09-Aug-2016)

Technical re-release of `v2.1.4`.

2.1.4 (01-Jul-2016)

The following bug fixes have been implemented:

- `tweakreg` crashes when run with a single input image and a reference catalog.
- Fixes an issue due to which `tweakreg`, when updating image headers, would not add '-SIP' suffix to CTYPE

2.1.3 (16-Mar-2016)

- Improved ASN input file handling.
- `astrodrizzle` does not delete `d2imfile` anylonger allowing multiple runs of `updatewcs` on the same WFPC2 image, see Ticket 1244 for more details.
- Allow exclusion regions in `tweakreg` to be in a different directory and allow relative path in exclusion region file name.
- Improved handling of empty input image lists.
- `tweakreg` bug fix: use absolute value of polygon area.

2.1.2 (12-Jan-2016)

- `runastrodriz` moved to `drizzlepac` from `acstools` and `wfc3tools` packages.
- Improved logic for duplicate input detection.
- Improved logic for handling custom WCS parameters in `astrodrizzle`.
- Compatibility improvements with Python 3.

2.1.1

Available under SSBX/IRAFX starting: Nov 17, 2015

This release includes the following bug fixes:

- Resolved order of operation problems when processing WFPC2 data with DGEOFILEs.
- The conversion of the WFPC2 `DGEOFILE` into `D2IMFILE` is now incorporated into `STWCS` v1.2.3 (r47112, r47113, r47114) rather than a part of `astrodrizzle`. This requires users to run updatewcs first, then `astrodrizzle`/`tweakreg` will work with that WFPC2 data seamlessly (as if they were ACS or WFC3 data).
- Compatibility improvements with Python 3.

2.1.0

Available under SSBX/IRAFX starting: Nov 2, 2015

This version builds upon the major set of changes implemented in v2.0.0 by not only fixing some bugs, but also cleaning up/changing/revising some APIs and docstrings. The complete list of changes includes:

- [API Change] The 'updatewcs' parameter was removed from both the `astrodrizzle` and `tweakreg` interactive TEAL interfaces. The 'updatewcs' parameter can still be used with the Python interface for both the `astrodrizzle`. `astrodrizzle``() and ``tweakreg`. Call the `stwcs.updatewcs.updatewcs()` function separately before running `astrodrizzle` or `tweakreg`.

- [API Change] The stand-alone interface for the blot routine (`ablot.blot()`) has been revised to work seamlessly with astrodrizzle-generated products while being more obvious how to call it correctly. The help file for this task was also heavily revised to document all the input parameters and to provide an example of how to use the task.
- [API Change] Coordinate transformation task (`pixtopix`/`pixtosky`/`skytopix`) interfaces changed to be more consistent, yet remain backward-compatible for now.
- Both `astrodrizzle` and `tweakreg` now return an output CD matrix which has identical cross-terms indicating the same scale and orientation in each axis (an orthogonal CD matrix). This relies on a revision to the `stwcs.distortion.utils.output_wcs()` function.
- The user interfaces to all 3 coordinate transformation tasks now use 'coordfile' as the input file of coordinates to transform. The use of 'coords' has been deprecated, but still can be used if needed. However, use of 'coordfile' will always override any input provided simultaneously with 'coords' parameter. Help files have been updated to document this as clearly as possible for users.
- User-provided list of input catalogs no longer needs to be matched exactly with input files. As long as all input images are included in input catalog list in any order, `tweakreg` will apply the correct catalog to the correct file.
- `tweakreg` has been updated to correctly and fully apply source selection criteria for both input source catalogs and reference source catalogs based on `fluxmin`, `fluxmax` and `nbright` for each.
- All use of keyword deletion has been updated in `drizzlepac` (and `fitsblender`) to avoid warnings from astropy.
- All 3 coordinate transformation tasks rely on the input of valid WCS information for the calculations. These tasks now warn the user when it could not find a valid WCS and instead defaulted to using a unity WCS, so that the user can understand what input needs to be checked/revised to get the correct results.
- Exclusion/inclusion region files that can be used with `tweakreg` can now be specified in image coordinates and sky coordinates and will only support files written out using DS9-compatible format.
- The filename for 'final_refimage' in `astrodrizzle` and 'refimage' in `tweakreg` can now be specified with OR without an extension, such as '[sci,1]' or '[0]'. If no extension is specified, it will automatically look for the first extension with a valid HSTWCS and use that. This makes the use of this parameter in both place consistent and more general than before.
- The reported fit as written out to a file has been slightly modified to report more appropriate numbers of significant digits for the results.
- Use of astrolib.coords was removed from `drizzlepac` and replaced by use of astropy functions instead. This eliminated one more obsolete dependency in our software.
- Code was revised to rely entirely on `astropy.wcs` instead of stand-alone pywcs.
- Code was revised to rely entirely on `astropy.io.fits` instead of stand-alone pyfits.
- Added `photeq` task to account for inverse sensitivity variations across detector chips and/or epochs.
- WFPC2 data from the archive with `DGEOFILE` reference files will now need to be processed using `stwcs.updatewcs` before running them through `astrodrizzle` or `tweakreg`. This update converts the obsolete, unsupported `DGEOFILE` correction for the WFPC2 data into a `D2IMFILE` specific for each

WFPC2 observation, then uses that to convert the WCS based on the new conventions used for ACS and WFC3.

This set of changes represents the last major development effort for `DrizzlePac` in support of HST. Support of this code will continue throughout the lifetime of HST, but will be limited primarily to bug fixes to keep the code viable as Python libraries used by `DrizzlePac` continue to develop and evolve with the language.

2.0.0

** Available under SSBX/IRAFX starting:** Aug 4, 2014

This version encompasses a large number of updates and revisions to the `DrizzlePac` code, including the addition of new tasks and several parameter name changes. The scope of these changes indicates the level of effort that went into improving the `DrizzlePac` code to make it easier and more productive for users. The most significant updates to the `DrizzlePac` code include:

- The Python code has been updated to work identically (without change) under both Python 2.7 and Python 3.x.
- Implementing sky matching, a new algorithm for matching the sky across a set of images being combined by `astrodrizzle`.
- Updating `tweakreg` to now align full mosaics where some images may not overlap others in the mosaic.
- Added the option to write out single drizzle step images as compressed images (to save disk space for large mosaics, and I/O time for single drizzle step).
- Improved `tweakreg` residual plots visually while allowing them to be written out automatically when `tweakreg` gets run in non-interactive mode.
- Renamed parameters in `tweakreg` and imagefind to eliminate name clashes.
- Added option to select sources based on sharpness/roundness when `tweakreg` searches for sources.
- Added support for exclusion and inclusion regions arbitrary shape/size when `tweakreg` searches for sources.
- Added a full set of source detection parameters for reference image to support multi-instrument alignment in `tweakreg`.
- Added support for new (simpler, more robust) ACS calibration of time-dependent distortion.
- A full 6-parameter general linear fit can now be performed using `tweakreg`, in addition to shift and rscale.
- Cleaned up logic for sky-subtraction: user can now turn off sky-subtraction with skysub=no, and still specify a user-defined sky value as the skyuser keyword. This will reduce(eliminate?) the need to manually set `MDRIZSKY=0`.

In addition to these major updates/changes, numerous smaller bugs were fixed and other revisions were implemented which affected a small portion of the use cases, such as:

- headerlet code now accepts lists of files to be updated.

- source sky positions (RA and Dec) now included in match file.
- DQ flags can now be taken into account when performing source finding in `tweakreg`.
- all intermediate files generated by `astrodrizzle` will now be removed when using 'clean'='yes'.
- a problem was fixed that caused `createMedian` to crash where there were no good pixels in one of the images (when they did not overlap).
- interpretation of shiftfile now improved to handle arbitrarily-long filenames, rather than being limited to 24 character filenames.
- documentation has been updated, sometimes with a lot more extensive descriptions.

This version of `DrizzlePac` also requires use of the latest release version of astropy primarily for WCS and FITS I/O support.

1.1.16

Publicly Released through PyPI: Mar 27, 2014

Available under SSBX/IRAFX starting: Mar 13, 2014

- Support for WFPC2 GEIS input images improved to correctly find the associated DQ images.
- Static mask files created for all chips in an image now get deleted when using the 'group' parameter to only drizzle a single chip or subset of chips.
- Fixed problem caused by changes to `stsci.tools` code so that `drizzlepac` will reference the correct extensions in input images.

1.1.15 (30-Dec-2013)

Publicly Released through PyPI: Jan 14, 2014

Available under SSBX/IRAFX starting: Jan 6, 2014

Bug fixes

- Files created or updated by `drizzlepac`, `fitsblender`, or `STWCS` tasks, e.g. `tweakreg` or `apply_headerlet`, will now ensure that the `NEXTEND` keyword value correctly reflects the number of extensions in the FITS file upon completion.

1.1.14dev (21-Oct-2013)

Installed in OPUS: Dec 11, 2013

Available starting: Oct 28, 2013

Bug fixes

- DQ arrays in input images now get updated with cosmic-ray masks computed by `astrodrizzle` when run with the parameter `in_memory=True`. This restored the cosmic-ray masks detected during pipeline processing.

v1.1.13dev (11-Oct-2013)

available starting: Oct 21, 2013

- `tweakreg` can now be run in 'batch' mode. This allows the user to generate plots and have them saved to disk automatically without stopping processing and requiring any user input.

1.1.12dev (05-Sep-2013)

available starting: Sept 9, 2013

This version fixed a couple of bugs in `astrodrizzle`; namely,

- Logic was updated to support pixfrac = 0.0 without crashing. Ths code will now automatically reset the kernel to 'point' in that case.
- `astrodrizzle` now forcibly removes all OPUS WCS keywords from drizzle product headers.
- Default rules for generating drizzle product headers (as used in the archive) were modified to add definitions for 'float_one', 'int_one', 'zero' that generate output values of 1.0, 1, and 0 (zero) respectively for use as keyword values. This allows the LTM* rules to replace 'first' with 'float_one' so that the physical and image coordinates for drizzle products are consistent.

Additionally, changes were made to `STWCS` for reprocessing use:

- Problems with using `apply_headerlet_as_primary()` from the `STWCS` package on WFPC2 data have been corrected in this revision.

1.1.11dev (05-Jul-2013)

Available starting: July 15, 2013

- AstroDrizzle now can process all STIS data without crashing.

1.1.10dev (06-Feb-2013)

available starting: May 6, 2013

- The output drizzle image header no longer contains references to D2IM arrays. This allows `tweakreg` to work with drizzled images as input where 2-D D2IM corrections were needed.
- Deprecated references to PyFITS .has_key() methods were also removed from the entire package, making it compatible with PyFITS 3.2.x and later.

1.1.8dev (06-Feb-2013)

available starting: Feb 11, 2013

- Fixed a bug in `astrodrizzle` which caused blot to raise an exception when using 'sinc' interpolation.
- Cleaned up the logic for writing out the results from the pixtopix, pixtosky, and skytopix tasks to avoid an Exception when a list of inputs are provided and no output file is specified.
- A new parameter was added to the tweakback task to allow a user to specify the value of `WCSNAME` when updating the FLT images with a new solution from a DRZ image header.
- Code in tweakback for updating the header with a new WCS will now automatically generate a unique `WCSNAME` if the there is a WCS solution in the FLT headers with the default or user-defined value of `WCSNAME`.

1.1.7dev (18-Dec-2012)

available starting: Feb 4, 2013

- Updated astrodrizzle to work with input images which do not have `WCSNAME` defined. This should make it easier to support non-HST input images in the future.
- cleared up confusion between flux parameters in imagefindpars and catalog inputs in `tweakreg`.
- turned of use of fluxes for trimming input source catalogs when no flux column can be found in input source catalogs.

1.1.7dev (18-Dec-2012)

available starting: Dec 10, 2012

- Update `tweakreg` 2d histogram building mode to correctly find the peak when all the inputs match with the same offset (no spurious sources in either source catalog).
- Fixed a bug so that Ctrl-C does not cause an exception when used while `tweakreg` is running.
- revised the source finding logic to ignore sources near the image edge, a change from how daofind works (daofind expands the image with blanks then fits anyway).

- created a new function to apply the nsigma separation criteria to (try to) eliminate duplicate entries for the same source from the source list. It turns out daofind does have problems with reporting some duplicate sources as well. This function does not work perfectly, but works to remove nearly all (if not all) duplicates in most cases.

1.1.7dev (8-Jan-2012)

available starting: Jan 14, 2013

- Bug fixed in updatehdr module to allow shiftfiles without RMS columns to work as inputs to manually apply shifts to headers of input images.
- Revised `astrodrizzle` to update WCS of all input images BEFORE checking whether or not they are valid. This ensures that all files provided as input to `astrodrizzle` in the pipeline have the headers updated with the distortion model and new WCS.
- Images with NGOODPIX=0 now identified for WFC3 and WFPC2 inputs, so they can be ignored during `astrodrizzle` processing.
- Replaced 2d histogram building code originally written in Python with a C function that run about 4x faster.

1.1.6dev (5-Dec-2012)

available starting: Dec 10, 2012

- `tweakreg` v1.1.0 source finding algorithm now runs many times faster (no algorithmic changes). No changes have been made yet to speed up the 2d histogram source matching code.
- The 'pixtopix' task was updated to make the 'outimage' parameter optional by using the input image as the default. This required no API changes, but the help files were updated.
- Very minor update to guard against MDRIZTAB being specified without any explicit path.
- Update `astrodrizzle` to correctly report the exposure time, exposure start, and exposure end for the single drizzle products, in addition to insuring the final drizzle values remain correct.
- `astrodrizzle` also includes initial changes to safeguard the C code from getting improperly cast values from the configObj(TEAL) input.

1.1.5dev (23-Oct-2012)

available starting: Oct 29, 2012

- Scaling of sky array for WFC3/IR IVM generation now correct.
- template mask files for WFPC2 no longer generated so that WFPC2 data can now be processed using num_cores > 1 (parallel processing).
- interpretation of the 'group' parameter fixed to support a single integer, a comma-separated list of integers or a single 'sci,<n>' value. The values correspond to the FITS extension number of the extensions

that should be combined. This fix may also speed up the initialization step as more direct use of pyfits was implemented for the interpretation of the 'group' parameter.

1.1.1 (31-Aug-2012)

available starting: Sept 26, 2012

The HST Archive and operational calibration pipeline started using this version on Sept 26, 2012.

1.1.4dev (20-Sep-2012)

available starting: Sept 24, 2012

- Bug fixed to allow use of final_wht_type=IVM for processing WFPC2 data.
- Revised Initialization processing to speed it up by using more up-to-date, direct pyfits calls.

1.1.3 (7-Sep-2012)

available starting: Sept 17, 2012

- Fixed the logic so that crclean images always get created regardless of the value of the 'clean' parameter.

1.1.2 (5-Sep-2012)

available starting: Sept 10, 2012

- Remove the restriction of only being able to process images which have `WCSNAME` keyword as imposed by r15631. The removal of this restriction will now allow for processing of non-updated input files with `updatewcs=False` for cases where no distortion model exists for the data (as required by CADC).
- Added log statements reporting what sky value was actually used in the drizzle and blot steps

1.1.1 (30-Aug-2012)

available starting: Sept 3, 2012

- Major revision to `astrodrizzle` allowing the option to process without writing out any intermediate products to disk. The intermediate products remain in memory requiring significantly more memory than usual. This improves the overall processing time by eliminating as much disk activity as possible as long as the OS does not start disk swapping due to lack of RAM.
- revised to turn off 'updatewcs' when coeffs=False(no) so that exposures with filter combinations not found in the IDCTAB will not cause an error.

1.0.7 (21-Aug-2012)

available starting: Aug 27, 2012

- Fixes problems with missing single_sci images.
- Static mask step revised to skip updates to static mask if all pixel data falls within a single histogram bin. This avoids problems with masking out entire images, which happens if low S/N SBC data is processed with `static_mask=yes`.

1.0.6 (14-Aug-2012)

available starting: Aug 20, 2012

Use of IVM for final_wht now correct, as previous code used wrong inputs when IVM weighting was automatically generated by `astrodrizzle`.

1.0.5 (8-Aug-2012)

available starting: Aug 13, 2012

- Completely removed the use of the TIME arrays for weighting IR drizzle products so that the photometry for saturated sources in drizzled products now comes out correct.
- Corrected a problem with `astrodrizzle` which affected processing of WFPC2 data where CRPIX2 was not found when creating the output single sci image.

1.0.2 (13-July-2012)

available starting: Aug 3, 2012

The complete version of stsci_python can be downloaded from our download page

- stsci_python v2.13 Release Notes
- Old stsci_python release notes

1.0.1 (20-June-2012)

Used in archive/pipeline starting: July 10, 2012

Pipeline and archive started processing ACS data with this version.

1.0.0 (25-May-2012)

Used in archive/pipeline starting: June 6, 2012

Pipeline and archive first started using `astrodrizzle` by processing WFC3 images.

1.4 Citing DrizzlePac

For the design of AstroDrizzle and the enhancements to the FITS format it has introduced, please reference:

A.S. Fruchter, W. Hack, N. Dencheva, M. Droettboom, P. Greenfield, 2010, "BetaDrizzle: A Redesign of the MultiDrizzle Package" in STSCI Calibration Workshop Proceedings, Baltimore, MD, 21-23 July 2010, eds. Susana Deustua & Cristina Oliveira, Space Telescope Science Institute, pp 376 - 381.

```
@INPROCEEDINGS{2010bdrz.conf..382F,
       author = {{Fruchter}, A.~S. and {et al.}},
        title = "{BetaDrizzle: A Redesign of the MultiDrizzle Package}",
     keywords = {betadrizzle, drizzlepac, astrodrizzle, drizzle, dither},
    booktitle = {2010 Space Telescope Science Institute Calibration Workshop},
         year = 2010,
        month = jul,
        pages = {382-387},
       adsurl = {https://ui.adsabs.harvard.edu/abs/2010bdrz.conf..382F},
      adsnote = {Provided by the SAO/NASA Astrophysics Data System}
}
```

For the Drizzlepac Handbook:

Hoffmann, S. L., Mack, J., et al., 2021, "The DrizzlePac Handbook", Version 2.0, (Baltimore: STScI).

```
@INPROCEEDINGS{2021AAS...23821602H,
       author = {{Hoffmann}, S.~L. and {Mack}, J. and {Avila}, R. and {Martlin},␣
↪C. and {Cohen}, Y. and {Bajaj}, V.},
        title = "{New Drizzlepac Handbook Version 2.0 Released In Hdox: Updated␣
↪Documentation For HST Image Analysis}",
    booktitle = {American Astronomical Society Meeting Abstracts},
         year = 2021,
       series = {American Astronomical Society Meeting Abstracts},
       volume = {53},
        month = jun,
          eid = {216.02},
        pages = {216.02},
       adsurl = {https://ui.adsabs.harvard.edu/abs/2021AAS...23821602H},
      adsnote = {Provided by the SAO/NASA Astrophysics Data System}
}
```

CHAPTER

TWO

DRIZZLEPAC INTERFACE

DrizzlePac is written in a hierarchical manner with top-level code in python, and some lower-level code in C. The primary user interface for drizzling individual frames is the astrodrizzle.py `Astrodrizzle()` function, which accepts input parameters as variables or as a user-specified configuration (.cfg) file. Examples of how to use these functions can be found in the DrizzlePac Notebook Tutorials.

The main steps of the drizzling process are listed below. For more information on these steps please refer to the DrizzlePac Handbook. Additional information on the Hubble Advanced Product (HAP) Single- and Multi-visit mosaics can be found in the MAST Data Products section.

2.1 Primary User Interface: AstroDrizzle()

`AstroDrizzle` - Python implementation of `MultiDrizzle`

`AstroDrizzle` automates the process of aligning images in an output frame, identifying cosmic-rays, removing distortion, and then combining the images after removing the identified cosmic-rays.

This process involves a number of steps, such as:

- Processing the input images and input parameters
- Creating a static mask
- Performing sky subtraction
- Drizzling onto separate output images
- Creating the median image
- Blotting the median image
- Identifying and flagging cosmic-rays
- Final combination
- Cleaning-up of temporary files (when applicable)

A full description of this process can be found in the DrizzlePac Handbook.

The primary output from this task is the distortion-corrected, cosmic-ray cleaned, and combined image as a FITS file.

This task requires numerous user-settable parameters to control the primary aspects of each of the processing steps.

Authors
Warren Hack

License
License

drizzlepac.astrodrizzle.**AstroDrizzle**(*input=None*, *mdriztab=False*, *editpars=False*, *configobj=None*, *wcsmap=None*, ***input_dict*)

Parameters

input
[str or list of str (Default = `'*flt.fits'`)] The name or names of the input files to be processed, which can be provided in any of the following forms:

- filename of a single image
- filename of an association (`ASN`) table
- wild-card specification for files in directory
- comma-separated list of filenames
- `@file` filelist containing list of desired input filenames. The file list needs to be provided as an ASCII text file containing a list of filenames for all input images with one filename on each line of the file. If inverse variance maps (`IVM` maps) have also been created by the user and are to be used (by specifying `'IVM'` to the parameter `final_wht_type`), then these are simply provided as a second column in the filelist, with each `IVM` filename listed on the same line as a second entry, after its corresponding exposure filename.

Note: If the user specifies `IVM` for the `final_wht_type`, but does not provide the names of `IVM` files, `AstroDrizzle` will automatically generate the `IVM` files itself for each input exposure.

---

mdriztab
[bool (Default = False)] This button will immediately update the parameter values in the `TEAL` GUI based on those provided by the `MDRIZTAB` reference table referenced in the first input image. This requires that the `MDRIZTAB` reference file be available locally.

editpars
[bool (Default = False)] A parameter that allows user to edit input parameters by hand in the GUI. `True` to use the GUI to edit parameters.

configobj
[ConfigObjPars, ConfigObj, dict (Default = None)] An instance of `stsci.tools.cfgpars.ConfigObjPars` or `stsci.tools.configobj.ConfigObj`

which overrides default parameter settings. When `configobj` is `defaults`, default parameter values are loaded from the user local configuration file usually located in `~/.teal/astrodrizzle.cfg` or a matching configuration file in the current directory. This configuration file stores most recent settings that an user used when running `AstroDrizzle` through the TEAL interface. When `configobj` is `None`, `AstroDrizzle` parameters not provided explicitly will be initialized with their default values as described in the "Other Parameters" section.

wcsmap

[wcs_functions.WCSMap, None (Default = None)] An instance of `wcs_functions.WCSMap` which can be used to override the default `WCS` mapping for the input images. This parameter is used to specify the mapping between the input images and the output frame. If `None` is provided, then the default `WCS` mapping will be used.

input_dict

[dict, optional] An optional list of parameters specified by the user, which can also be used to override the defaults.

Note: This list of parameters **can** include the `updatewcs` parameter, even though this parameter no longer can be set through the `TEAL` GUI.

Note: This list of parameters **can** contain parameters specific to the `AstroDrizzle` task itself described here in the "Other Parameters" section.

Other Parameters

output

[str (Default = '')] The rootname for the output drizzled products. This step can result in the creation of several files, including:

- copies of each input image as a `FITS` image, if `workinplace='Yes'` and/or input images are in `GEIS` format.
- mask files and coeffs files created by `PyDrizzle` for use by `drizzle`.

If an association file has been given as input, the specified filename will be used instead of the product name specified in the ASN file. Similarly, if a single exposure is provided, the rootname of the single exposure will be used for the output product instead of relying on the input rootname. If no value is provided when a filelist or wild-card specification is given as input, then a rootname of `'final'` will be used for the output file name.

runfile

[str (Default = 'astrodrizzle.log')] This log file will contain all the output messages generated during processing, including full details of any errors/exceptions. These messages will be a super-set of those reported to the screen during processing.

wcskey

[str (Default = '')] This parameter corresponds to the *key* for the WCS being selected by the user. It allows the user to select which WCS solution should be used for processing the images when multiple WCS's have been updated in each input image header using the Paper I Multiple WCS FITS standard.

> **Warning:** Use of this parameter should be done only when all input images have been updated using the Paper I FITS standard for specifying Multiple WCS's in each image header. This parameter assumes that the same WCS letter corresponds to WCS's that have been updated in a consistent manner. For example, all input images have been updated to be consistent with their distortion model in the WCS's with key of *A*.

proc_unit

[str (Default = 'native')] The units to be used for the final output drizzled product. Valid values and definitions are:

- `'native'`: Output DRZ product and input 'values' given in the native units of the input image.
- `'electrons'`: Output DRZ product and input 'values' given in units of electrons.

coeffs

[bool (Default = Yes)] This parameter determines whether or not to use the coefficients stored in the each input image header. If turned off, no distortion coefficients will be applied during the coordinate transformations.

context

[bool (Default = Yes)] This parameter specifies whether or not to create a context image during the final drizzle combination. The context image contains the information regarding which image(s) contributed to each pixel encoded as a bitmask. More information on context images can be obtained from the ACS Data Handbook.

group

[int (Default = None)] This parameter establishes whether or not a single FITS extension, or group will be drizzled. If an extension is provided, then only that chip will be drizzled onto the output frame. Either a FITS extension number, a GEIS group number (such as '1'), or a FITS extension name (such as `'sci,1'`) may be specified.

build

[bool (Default = No)] When this parameter is set to `'Yes'` (`True`), AstroDrizzle will combine the separate 'drizzle' output files into a single multi-extension format FITS file. This combined output file will contain seperate SCI (science), WHT (weight), and CTX (context) extensions. If this parameter is set to `'No'` (`False`), a separate simple FITS file will be created for each aforementioned extension.

crbit

[int (Default = 4096)] This parameter sets the bit value for CR identification in the

DQ array.

stepsize
[int (Default = 10)] This parameter controls the internal grid of points used in the coordinate transformation from the input image to the output frame. The default value of 10 indicates that every 10th pixel will be transformed using the full `WCS`-based transformation. All remaining pixels will then be transformed using bilinear interpolation based on those pixels (i.e. every 10th pixel in the case of the default parameter setting) that were fully transformed.

resetbits
[int (Default = 4096)] This parameter allows the user to specify which DQ bits of each input image DQ array should be reset to a value of 0. This operation is performed on the copy of the input data after updating the headers based on the 'updatewcs' parameter, and prior to starting any of the `AstroDrizzle` processing steps (static mask, sky subtraction, and so on).

num_cores
[int (Default = None)] This specifies the number of CPU cores to use during processing. Any value less than 2 will disable all use of parallel processing. At this time, this parameter will be forced to a value of 1 internally when running under Windows. This restriction will be lifted in a future release once issues in the code related to using logging with multiprocessing are resolved.

in_memory
[bool (Default = False)] This parameter sets whether or not to keep all intermediate products in memory when processing. This includes all single drizzle products (`*single_sci` and `*single_wht`), median image, blot images, and crmask images. The use of this option will therefore require significantly more memory than usual to process the data while reducing the overall processing time by eliminating most of the disk activity. *Only* the products of the final drizzle step will get written out when this parameter gets specified as `True`.

rules_file
[str (Default = "")] Rules for how to blend the header keyword values for all the input exposures into a single header for the drizzle products are specified using this `rules_file`. The `fitsblender` package uses this file to determine what keywords should be written out to the drizzle product headers. If no file is specified (default), the rules file for the instrument as included with the `fitsblender` package will be used for defining the product headers.

****STATE OF INPUT FILES****
restore: bool (Default = No)
Setting this to `'Yes'` (`True`) directs `AstroDrizzle` to copy the input images from the `'OrIg_files'` sub-directory and use them for processing, if they had been archived by `AstroDrizzle` using the `preserve` or `overwrite` parameters already. If set to `'Yes'` and the input files had not been archived already, it will simply ignore this and work with the current input images.

preserve
[bool (Default = Yes)] Copy input files to archive directory, if not already archived.

This parameter determines whether or not AstroDrizzle creates a copy of the input file in a sub-directory called 'OrIg_files'. If a copy already exists in this directory, then the previously existing version will NOT be overwritten.

overwrite
[bool (Default = No)] Copy input files into archive, overwriting older files if required? This parameter will cause AstroDrizzle to make a copy of each input file in the 'OrIg_files' directory regardless of whether a previous copy existed or not, and will overwrite any previous copy should it be present.

clean
[bool (Default = No)] The temporary files created by AstroDrizzle can be automatically removed by setting this parameter to 'Yes' (True). The affected files include the coefficient and static mask files created by PyDrizzle, in addition to other intermediate files created by AstroDrizzle. It is often useful to retain the intermediate files and examine them when first learning how to run AstroDrizzle. However, when running AstroDrizzle routinely, or on a small disk drive, these files can be removed to conserve space.

STEP 1: STATIC MASK
static
[bool (Default = Yes)] Create a static bad-pixel mask from the data? This mask flags all pixels that deviate by more than a value of 'static_sig' sigma below the image median, since these pixels are typically the result of bad pixel oversubtraction in the dark image during calibration.

static_sig
[float (Default = 4.0)] The number of sigma below the RMS to use as the clipping limit for creating the static mask.

****STEP 2: SKY SUBTRACTION****
skysub
[bool (Default = Yes)] Turn on or off sky subtraction on the input data. When skysub is set to no, then skyuser field will be enabled and if user specifies a header keyword showing the sky value in the image, then that value will be used for CR-rejection but it will not be subtracted from the (drizzled) image data. If user sets skysub to yes then skyuser field will be disabled (and if it is not empty - it will be ignored) and user can use one of the methods available through the skymethod parameter to compute the sky or provide a file (see skyfile parameter) with values that should be subtracted from (single) drizzled images.

skymethod
[{'localmin', 'globalmin+match', 'globalmin', 'match'} (Default = 'localmin')] Select the algorithm for sky computation:

- **'localmin'**: compute a common sky for all members of *an exposure*. For a typical use, it will compute sky values for each chip/image extension (marked for sky subtraction in the input parameter) in an input image, and it will subtract the previously found minimum sky value from all chips (marked for sky subtraction) in that image. This process is repeated for each input image.

Note: This setting is recommended when regions of overlap between images are dominated by "pure" sky (as opposite to extended, diffuse sources).

Note: This is similar to the "skysub" algorithm used in previous versions of `AstroDrizzle`.

- `'globalmin'`: compute a common sky value for all members of **all** "skylines". It will compute sky values for each chip/image extension (marked for sky subtraction in the `input` parameter) in **all** input images, find the minimum sky value, and then it will subtract the **same** minimum sky value from **all** chips (marked for sky subtraction) in **all** images. This method *may* useful when input images already have matched background values.
- `'match'`: compute differences in sky values between images in common (pairwise) sky regions. In this case computed sky values will be relative (delta) to the sky computed in one of the input images whose sky value will be set to (reported to be) 0. This setting will "equalize" sky values between the images in large mosaics. However, this method is not recommended when used in conjunction with AstroDrizzle because it computes relative sky values while `AstroDrizzle` needs "measured" sky values for median image generation and CR rejection.
- `'globalmin+match'`: first find a minimum "global" sky value in all input images and then use `'match'` method to equalize sky values between images.

Note: This is the *recommended* setting for images containing diffuse sources (e.g., galaxies, nebulae) covering significant parts of the image.

skywidth
[float (Default = 0.3)] Bin width, in sigma, used to sample the distribution of pixel flux values in order to compute the sky background statistics.

skystat
[{'median', 'mode', 'mean'} (Default = 'median')] Statistical method for determining the sky value from the image pixel values.

skylower
[float (Default = None)] Lower limit of usable pixel values for computing the sky. This value should be specified in the units of the input image(s).

skyupper
[float (Default = None)] Upper limit of usable pixel values for computing the sky. This value should be specified in the units of the input image(s).

skyclip
[int (Default = 5)] Number of clipping iterations to use when computing the sky value.

skylsigma
[float (Default = 4.0)] Lower clipping limit, in sigma, used when computing the sky value.

skyusigma
[float (Default = 4.0)] Upper clipping limit, in sigma, used when computing the sky value.

skymask_cat
[str (Default = '')] File name of a catalog file listing user masks to be used with images.

use_static
[bool (Default = True)] Specifies whether or not to use static mask to exclude masked image pixels from sky computations.

sky_bits
[int, str, None (Default = 0)] Integer sum of all the DQ bit values from the input image's DQ array that should be considered "good" when building masks for sky computations. For example, if pixels in the DQ array can be combinations of 1, 2, 4, and 8 flags and one wants to consider DQ "defects" having flags 2 and 4 as being acceptable for sky computations, then `sky_bits` should be set to 2+4=6. Then a DQ pixel having values 2,4, or 6 will be considered a good pixel, while a DQ pixel with a value, e.g., 1+2=3, 4+8=12, etc. will be flagged as a "bad" pixel.

Alternatively, one can enter a comma- or '+'-separated list of integer bit flags that should be added to obtain the final "good" bits. For example, both `4,8` and `4+8` are equivalent to setting `sky_bits` to 12.

Default value (0) will make *all* non-zero pixels in the DQ mask to be considered "bad" pixels, and the corresponding image pixels will not be used for sky computations.

Set `sky_bits` to `None` to turn off the use of image's DQ array for sky computations.

In order to reverse the meaning of the `sky_bits` parameter from indicating values of the "good" DQ flags to indicating the "bad" DQ flags, prepend '~' to the string value. For example, in order not to use pixels with DQ flags 4 and 8 for sky computations and to consider as "good" all other pixels (regardless of their DQ flag), set `sky_bits` to `~4+8`, or `~4,8`. To obtain the same effect with an `int` input value (except for 0), enter -(4+8+1)=-13. Following this convention, a `sky_bits` string value of `'~0'` would be equivalent to setting `sky_bits=None`.

Note: DQ masks (if used), *will be* combined with user masks specified in the

input @-file.

Note: To summarize, below are provided allowable syntaxes for `sky_bits`:

- Specify that bits 4,8, and 512 be considered "good" bits and that all other bits be considered "bad" bits:
  - Integer: 524 [numerically: 4+8+512=524]
  - String: 4+8+512
  - String: 4,8,512
- Specify that only bits 4,8, and 512 be considered "bad" bits and that all other bits be considered "good" bits:
  - Integer: -525 [numerically: ~(4+8+512)=~524=-(524+1)=-525]
  - String: ~4+8+512 or ~(4+8+512)
  - String: ~4,8,512 or ~(4,8,512)

---

skyfile

[str (Default = '')] Name of file containing user-computed sky values to be used with each input image. This ASCII file should only contain 2 columns: image filename in column 1 and sky value in column 2 (and higher hor multi-chip images). The sky value should be provided in units that match the units of the input image and for multi-chip images, if only one sky value was provided in column 2, the same value will be applied to all chips. If more than one sky value are provided (in columns 2, 3, ...) then the number of sky values should match the number of `SCI` extensions in the images.

skyuser

[str (Default = '')] Name of header keyword which records the sky value already subtracted from the image by the user. The `skyuser` parameter is ignored when `skysub` is set to `yes`.

Alternatively, user can enter the name of a file that contains user-computed sky values. To distinguish a file name from a header keyword, prepend `'@'` to the file name. For example `'@my_sky_values.txt'`. The format of the file with user-supplied sky values is the same as that of a `skyfile`.

Note: When `skysub='no'` and `skyuser` field is empty, then `AstroDrizzle` will assume that sky background is 0.0 for the purpose of cosmic-ray rejection.

****STEP 3: DRIZZLE SEPARATE IMAGES****

driz_separate

[bool (Default = Yes)] This parameter specifies whether or not to drizzle each input image onto separate output images. The separate output images will all have the

same WCS as the final combined output frame. These images are used to create the median image, needed for cosmic ray rejection.

driz_sep_kernel
[str {'square', 'point', 'turbo', 'gaussian', 'lanczos3'} (Default = 'turbo')] Used for the initial separate drizzling operation only, this parameter specifies the form of the kernel function used to distribute flux onto the separate output images. The current options are:

- `'square'`: original classic drizzling kernel
- `'point'`: this kernel is a point so each input pixel can only contribute to the single pixel that is closest to the output position. It is equivalent to the limit as `pixfrac->0`, and is very fast.
- `'turbo'`: this is similar to `kernel='square'` but the box is always the same shape and size on the output grid, and is always aligned with the X and Y axes. This may result in a significant speed increase.
- `'gaussian'`: this kernel is a circular gaussian with a FWHM equal to the value of pixfrac, measured in input pixels.
- `'lanczos3'`: a Lanczos style kernel, extending a radius of 3 pixels from the center of the detection. The Lanczos kernel is a damped and bounded form of the "sinc" interpolator, and is very effective for resampling single images when `scale=pixfrac=1`. It leads to less resolution loss than other kernels, and typically results in reduced correlated noise in outputs.

> **Warning:** While the `'gaussian'` and `'lanczos3'` kernels may produce reasonable results and can be useful in certain cases, they do not conserve flux; understand the effects of these kernels before using them.

> **Warning:** The `'lanczos3'` kernel tends to result in much slower processing as compared to other kernel options. This option should never be used for `pixfrac!=1.0`, and is not recommended for `scale!=1.0`.

The default for this step is `'turbo'` since it is much faster than `'square'`, and it is quite satisfactory for the purposes of generating the median image. More information about the different kernels can be found in the help file for the drizzle task.

driz_sep_wt_scl
[float (Default = exptime)] This parameter specifies the weighting factor for input image. If `driz_sep_wt_scl=exptime`, then the scaling value will be set equal to the exposure time found in the image header. The use of the default value is recommended for producing optimal behavior for most scenarious. It is possible to set `wt_scl=expsq` for weighting by the square of the exposure time, which is optimal for read-noise dominated images.

driz_sep_pixfrac
[float (Default = 1.0)] Fraction by which input pixels are "shrunk" before being drizzled onto the output image grid, given as a real number between 0 and 1. This specifies the size of the footprint, or "dropsize", of a pixel in units of the input pixel size. If pixfrac is set to less than 0.001, the kernel parameter will be reset to `'point'` for more efficient processing. In the step of drizzling each input image onto a separate output image, the default value of 1.0 is best in order to ensure that each output drizzled image is fully populated with pixels from the input image. For more information, see the help for the `drizzle` task.

driz_sep_fillval
[int (Default = None)] Value to be assigned to output pixels that have zero weight, or that receive flux from any input pixels during drizzling. This parameter corresponds to the `fillval` parameter of the 'drizzle' task. If the default of `None` is used, and if the weight in both the input and output images for a given pixel are zero, then the output pixel will be set to the value it would have had if the input had a non-zero weight. Otherwise, if a numerical value is provided (e.g. 0), then these pixels will be set to that value.

driz_sep_bits
[int, str, or None (Default = 0)] Integer sum of all the DQ bit values from the input image's DQ array that should be considered 'good' when building the weighting mask. This can also be used to reset pixels to good if they had been flagged as cosmic rays during a previous run of `AstroDrizzle`, by adding the value 4096 for `ACS` and `WFPC2` data. For possible input formats, see the description for `sky_bits` parameter.

****STEP 3a: CUSTOM WCS FOR SEPARATE OUTPUTS****
driz_sep_wcs
[bool (Default = No)] Define custom `WCS` for seperate output images?

driz_sep_refimage
[str (Default = '')] Reference image from which a `WCS` solution can be obtained.

driz_sep_rot
[float (Default = None)] Position Angle of output image's Y-axis relative to North. A value of 0.0 would orient the final output image to be North up. The default of `None` specifies that the images will not be rotated, but will instead be drizzled in the default orientation for the camera with the x and y axes of the drizzled image corresponding approximately to the detector axes. This conserves disk space, as these single drizzled images are only used in the intermediate step of creating a median image.

driz_sep_scale
[float (Default = None)] Linear size of the output pixels in arcseconds/pixel for each separate drizzled image (used in creating the median for cosmic ray rejection). The default value of None specifies that the undistorted pixel scale for the first input image will be used as the pixel scale for all the output images.

driz_sep_outnx
[int (Default = None)] Size, in pixels, of the X axis in the output images that each

input will be drizzled onto. If no value is specified, the smallest size that can accommodate the full dithered field will be used.

driz_sep_outny
[int (Default = None)] Size, in pixels, of the Y axis in the output images that each input will be drizzled onto. If no value is specified, the smallest size that can accommodate the full dithered field will be used.

driz_sep_ra
[float (Default = None)] Right ascension (in decimal degrees) specifying the center of the output image. If this value is not designated, the center will automatically be calculated based on the distribution of image dither positions.

driz_sep_dec
[float (Default = None)] Declination (in decimal degrees) specifying the center of the output image. If this value is not designated, the center will automatically be calculated based on the distribution of image dither positions.

****STEP 4: CREATE MEDIAN IMAGE****
median
[bool (Default = Yes)] This parameter specifies whether or not to create a median image. This median image will be used as the comparison 'truth' image in the cosmic ray rejection step.

median_newmasks
[bool (Default = Yes)] This parameter specifies whether or not new mask files will be created when the median image is created. These masks are generated from weight files previously produced by the "driz_separate" step, and contain all bad pixel information used to exclude pixels when calculating the median. Generally this step should be set to `'Yes'` (`True`), unless for some reason, it is desirable to include bad pixel information when generating the median.

combine_maskpt
[float (Default = 0.3)] Percentage of weight image values, below which the are flagged.

combine_type
[str { 'median', 'mean', 'minmed', 'imedian', 'imean', 'iminmed' } (Default = 'minmed')] This parameter defines the method that will be used to create the median image. The 'mean' and 'median' options set the calculation type when running 'numcombine', a numpy method for median-combining arrays to create the median image. The `'minmed'` option will produce an image that is generally the same as the median, except in cases where the median is significantly higher than the minimum good pixel value. In this case, `'minmed'` will choose the minimum value. The sigma thresholds for this decision are provided by the `'combine_nsigma'` parameter. However, as the `'combine_nsigma'` parameter does not adjust for the larger probability of a single "nsigma" event with a greater number of images, `'minmed'` will bias the comparison image low for a large number of images. The value of sigma is computed as $\sigma = \sqrt{(M + S + R^2)}$, where *M* is the median image data (in electrons), *S* is the value of the subtracted sky (in electrons), and *R* is the value of the readout noise (in electrons). `'minmed'` is highly recommended

for three images, and is good for four to six images, but should be avoided for ten or more images.

A value of 'median' is the recommended method for a large number of images, and works equally well as minmed down to approximately four images. However, the user should set the `combine_nhigh` parameter to a value of 1 when using "median" with four images, and consider raising this parameter's value for larger numbers of images. As a median averages the two inner values when the number of values being considered is even, the user may want to keep the total number of images minus `combine_nhigh` odd when using `median`.

The options starting with `'i'`, such as `'imedian'`, works just like the normal median operation except when dealing with a pixel were all the values are flagged as 'bad'. In this case, the `'i'` functions return the last pixel in the stack as if it were good. This will prevent saturated pixels in the image from leaving holes in the middle of the stars, for example.

combine_nsigma
[float (Default = '4 3')] This parameter defines the sigmas used for accepting minimum values, rather than median values, when using the `'minmed'` combination method. If two values are specified the first value will be used in the initial choice between median and minimum, while the second value will be used in the "growing" step to reject additional pixels around those identified in the first step. If only one value is specified, then it is used in both steps.

combine_nlow
[int (Default = 0)] This parameter sets the number of low value pixels to reject automatically during image combination.

combine_nhigh
[int (Default = 0)] This parameter sets the number of high value pixels to reject automatically during image combination.

combine_lthresh
[float (Default = None)] Sets the lower threshold for clipping input pixel values during image combination. This value gets passed directly to `imcombine` for use in creating the median image. If the parameter is set to `None`, no thresholds will be imposed.

combine_hthresh
[float (Default = None)] This parameter sets the upper threshold for clipping input pixel values during image combination. The value for this parameter is passed directly to `imcombine` for use in creating the median image. If the parameter is set to `None`, no thresholds will be imposed.

combine_grow
[int (Default = 1)] Width, in pixels, beyond the limit set by the rejection algorithm being used, for additional pixels to be rejected in an image. This parameter is used to set the `grow` parameter in `imcombine` for use in creating the median image **only when** `combine_type` is `'(i)minmed'`. When `combine_type` is anything other than `'(i)minmed'`, this parameter is ignored (set to 0).

combine_bufsize
[float (Default = None)] Size of buffer, in MB (MiB), to use when reading in each section of each input image. The default buffer size is 1MB. The larger the buffer size, the fewer times the code needs to open each input image and the more memory will be required to create the median image. A larger buffer can be helpful when using compression, since slower copies need to be made of each set of rows from each input image instead of using memory-mapping.

****STEP 5: BLOT BACK THE MEDIAN IMAGE****
blot
[bool (Default = Yes)] Perform the blot operation on the median image? If set to `'Yes'` (`True`), the output will be median smoothed images that match each input chips location, and will be used in the cosmic ray rejection step.

blot_interp
[str{'nearest', 'linear', 'poly3', 'poly5', 'sinc'} (Default = 'poly5')] This parameter defines the method of interpolation to be used when blotting drizzled images back to their original `WCS` solution. Valid options include:

- `'nearest'`: Nearest neighbor
- `'linear'`: Bilinear interpolation in x and y
- `'poly3'`: Third order interior polynomial in x and y
- `'poly5'`: Fifth order interior polynomial in x and y
- `'sinc'`: Sinc interpolation (accurate but slow)

The `'poly5'` interpolation method has been chosen as the default because it is relatively fast and accurate.

If `'sinc'` interpolation is selected, then the value of the parameter for `blot_sinscl` will be used to specify the size of the sinc interpolation kernel.

blot_sinscl
[float (Default = 1.0)] Size of the sinc interpolation kernel in pixels.

blot_addsky
[bool (Default = Yes)] Add back a sky value using the MDRIZSKY value from the header. If `'Yes'` (`True`), the blot_skyval parameter is ignored.

blot_skyval
[float (Default = 0.0)] This is a user-specified custom sky value to be added to the blot image. This is only used if blot_addsky is `'No'` (`False`).

****STEP 6: REMOVE COSMIC RAYS WITH DERIV, DRIZ_CR****
driz_cr
[bool (Default = Yes)] Perform cosmic-ray detection? If set to `'Yes'` (`True`), cosmic-rays will be detected and used to create cosmic-ray masks based on the algorithms from 'deriv' and `driz_cr`.

driz_cr_corr
[bool (Default = No)] Create a cosmic-ray cleaned input image? If set to `'Yes'` (`True`), a cosmic-ray cleaned `_crclean` image will be generated directly from

the input image, and a corresponding _crmask file will be written to document detected pixels affected by cosmic-rays.

driz_cr_snr
[list of floats (Default = '3.5 3.0')] The values for this parameter specify the signal-to-noise ratios for the `driz_cr` task to be used in detecting cosmic rays. See the help file for `driz_cr` for further discussion of this parameter.

driz_cr_grow
[int (Default = 1)] The radius, in pixels, around each detected cosmic-ray, in which more stringent detection criteria for additional cosmic rays will be used.

driz_cr_ctegrow
[int (Default = 0)] Length, in pixels, of the CTE tail that should be masked in the drizzled output.

driz_cr_scale
[str (Default = '1.2 0.7')] Scaling factor applied to the derivative in `driz_cr` when detecting cosmic-rays. See the help file for `driz_cr` for further discussion of this parameter.

****STEP 7: DRIZZLE FINAL COMBINED IMAGE****
driz_combine
[bool (Default = Yes)] This parameter specifies whether or not to drizzle each input image onto the final output image. This applies the generated cosmic-ray masks to the input images and creates a final, cleaned, distortion-corrected image.

final_wht_type
[{'EXP', 'ERR', 'IVM'} (Default = 'EXP')] Specify the type of weighting image to apply with the bad pixel mask for the final drizzle step. The options for this parameter include:

- `'EXP'`: The default of `'EXP'` indicates that the images will be weighted according to their exposure time, which is the standard behavior for drizzle. This weighting is a good approximation in the regime where the noise is dominated by photon counts from the sources, while contributions from sky background, read-noise and dark current are negligible. This option is provided as the default since it produces reliable weighting for all types of data, including older instruments (eg., `WFPC2`), where more sophisticated options may not be available.
- `'ERR'`: Specifying `'ERR'` is an alternative for `ACS` and `STIS` data. In these cases, the final drizzled images will be weighted according to the inverse variance of each pixel in the input exposure files, calculated from the error array data extension that is in each calibrated input exposure file. This array is exposure time dependent, and encapsulates all of the noise sources in each exposure including read-noise, dark current, sky background, and Poisson noise from the sources themselves. For `WFPC2`, the `ERR` array is not produced during the calibration process, and therefore is not a viable option. We advise extreme caution when selecting the `'ERR'` option, since the nature of this weighting scheme can introduce photometric discrepancies in sharp unresolved sources, although these effects are minimized for sources with gradual variations between pixels.

The "EXP" weighting option does not suffer from these effects, and is therefore the recommended option.

- `'IVM'`: Specifying `'IVM'` allows the user to either supply their own inverse-variance weighting map, or allow `AstroDrizzle` to generate one automatically on-the-fly during the final drizzle step. This parameter option may be necessary for specific purposes. For example, to create a drizzled weight file for software such as `SExtractor`, it is expected that a weight image containing all of the background noise sources (sky level, read-noise, dark current, etc), but not the Poisson noise from the objects themselves will be available. The user can create the inverse variance images and then specify their names using the `input` parameter for `AstroDrizzle` to specify an '@file'. This would be a single `ASCII` file containing the list of input calibrated exposure filenames (one per line), with a second column containing the name of the `IVM` file corresponding to each calibrated exposure. Each `IVM` file must have the same file format as the input file, and if provided as multi-extension `FITS` files (e.g., `ACS` or `STIS` data) then the `IVM` extension must have the `EXTNAME` of `'IVM'`. If no `IVM` files are specified on input, then `AstroDrizzle` will rely on the flat-field reference file and computed dark value from the image header to automatically generate an `IVM` file specific to each exposure.

final_kernel

[{'square', 'point', 'turbo', 'gaussian', 'lanczos3'} (Default = 'square')] This parameter specifies the form of the kernel function used to distribute flux onto the separate output images, for the initial separate drizzling operation only. The value options for this parameter include:

- `'square'`: original classic drizzling kernel
- `'point'`: this kernel is a point so each input pixel can only contribute to the single pixel that is closest to the output position. It is equivalent to the limit as `pixfrac->0`, and is very fast.
- `'gaussian'`: this kernel is a circular gaussian, measured in input pixels, with a FWHM value equal to the value of `pixfrac`.
- `'turbo'`: this is similar to kernel="square", except that the box is always the same shape and size on the output grid, and is always aligned with the `X` and `Y` axes. This may result in a significant speed increase.
- `'lanczos3'`: a Lanczos style kernel, extending a radius of 3 pixels from the center of the detection. The Lanczos kernel is a damped and bounded form of the "sinc" interpolator, and is very effective for resampling single images when `scale=pixfrac=1`. It leads to less resolution loss than other kernels, and typically results in reduced correlated noise in outputs.

> **Warning:** The `'lanczos3'` kernel tends to result in much slower processing as compared to other kernel options. This option should never be used for pixfrac != 1.0, and is not recommended for `scale!=1.0`.

The default for this step is `'turbo'` since it is much faster than `'square'`, and it is quite satisfactory for the purposes of generating the median image. More information about the different kernels can be found in the help file for the drizzle task.

final_wt_scl

[float (Default = exptime)] This parameter specifies the weighting factor for input image. If `final_wt_scl=exptime`, then the scaling value will be set equal to the exposure time found in the image header. The use of the default value is recommended for producing optimal behavior for most scenarious. It is possible to set `wt_scl='expsq'` for weighting by the square of the exposure time, which is optimal for read-noise dominated images.

final_pixfrac

[float (Default = 1.0)] Fraction by which input pixels are "shrunk" before being drizzled onto the output image grid, given as a real number between 0 and 1. This specifies the size of the footprint, or "dropsize", of a pixel in units of the input pixel size. If pixfrac is set to less than 0.001, the kernel parameter will be reset to `'point'` for more efficient processing. In the step of drizzling each input image onto a separate output image, the default value of 1.0 is best in order to ensure that each output drizzled image is fully populated with pixels from the input image. For more information, see the help for the 'drizzle' task.

final_fillval

[float (Default = None)] The value for this parameter is to be assigned to the output pixels that have zero weight or which do not receive flux from any input pixels during drizzling. This parameter corresponds to the `fillval` parameter of the `drizzle` task. If the default of `None` is used, and if the weight in both the input and output images for a given pixel are zero, then the output pixel will be set to the value it would have had if the input had a non-zero weight. Otherwise, if a numerical value is provided (e.g. 0), then these pixels will be set to that numerical value.

final_bits

[int, str, or None (Default = 0)] Integer sum for all of the `DQ` bit values from the input image's `DQ` array that should be considered 'good' when building the weight mask. This can also be used to reset pixels to good if they had been flagged as cosmic rays during a previous run of `AstroDrizzle`, by adding the value 4096 for `ACS` and `WFPC2` data. For possible input formats, see the description for `sky_bits` parameter.

final_units

[str (Default = 'cps')] This parameter determines the units of the final drizzle-combined image, and can either be `'counts'` or `'cps'`. It is passed through to `drizzle` in the final drizzle step.

****STEP 7a: CUSTOM WCS FOR FINAL OUTPUT****

final_wcs

[bool (Default = No)] Obtain the `WCS` solution from a user-designated reference image?

final_refimage
[str (Default = '')] Reference image from which a `WCS` solution can be obtained. If no extension is specified (such as 'sci,1' or '4'), then `AstroDrizzle` will automatically look for the **first** extension which contains a valid `HSTWCS` object to read in as the `WCS`. Otherwise, the user can explicitly provide the extension name for multi-extension `FITS` files.

final_rot
[float (Default = None)] Position Angle of output image's Y-axis relative to North. A value of 0.0 would orient the final output image to be North up. The default of `None` specifies that the images will not be rotated, but will instead be drizzled in the default orientation for the camera with the x and y axes of the drizzled image corresponding approximately to the detector axes. This conserves disk space, as these single drizzled images are only used in the intermediate step of creating a median image.

final_scale
[float (Default = None)] Linear size of the output pixels in arcseconds/pixel for each separate drizzled image (used in creating the median for cosmic ray rejection). The default value of None specifies that the undistorted pixel scale for the first input image will be used as the pixel scale for all the output images.

final_outnx
[int (Default = None)] Size, in pixels, of the X axis in the output images that each input will be drizzled onto. If no value is specified, the smallest size that can accommodate the full dithered field will be used.

final_outny
[int (Default = None)] Size, in pixels, of the Y axis in the output images that each input will be drizzled onto. If no value is specified, the smallest size that can accommodate the full dithered field will be used.

final_ra
[float (Default = None)] Right ascension (in decimal degrees) specifying the center of the output image. If this value is not designated, the center will automatically be calculated based on the distribution of image dither positions.

final_dec
[float (Default = None)] Declination (in decimal degrees) specifying the center of the output image. If this value is not designated, the center will automatically be calculated based on the distribution of image dither positions.

****INSTRUMENT PARAMETERS****
gain
[float (Default = None)] Value used to override instrument specific default gain values. The value is assumed to be in units of electrons/count. This parameter should not be populated if the `gainkeyword` parameter is in use.

gainkeyword
[str (Default = '')] Keyword used to specify a value, which is used to override the instrument specific default gain values. The value is assumed to be in units of

electrons/count. This parameter should not be populated if the `gain` parameter is in use.

rdnoise
[float (Default = None)] Value used to override instrument specific default readnoise values. The value is assumed to be in units of electrons. This parameter should not be populated if the `rnkeyword` parameter is in use.

rnkeyword
[str (Default = '')] Keyword used to specify a value, which is used to override the instrument specific default readnoise values. The value is assumed to be in units of electrons. This parameter should not be populated if the `rdnoise` parameter is in use.

exptime
[float (Default = None)] Value used to override default exposure time image header values. The value is assumed to be in units of seconds. This parameter should not be populated if the `expkeyword` parameter is in use.

expkeyword
[str (Default = '')] Keyword used to specify a value, which is used to override the default exposure time image header values. The value is assumed to be in units of seconds. This parameter should not be populated if the `exptime` parameter is in use.

****ADVANCED PARAMETERS AVAILABLE FROM COMMAND LINE****
updatewcs
[bool (Default = No)] This parameter specifies whether the `WCS` keywords are to be updated by running updatewcs on the input data, or left alone. The update performed by updatewcs not only recomputes the `WCS` based on the currently used `IDCTAB`, but also populates the header with the `SIP` coefficients. For `ACS/WFC` images, the time-dependence correction will also be applied to the `WCS` and `SIP` keywords. This parameter should be set to `'No'` (`False`) when the `WCS` keywords have been carefully set by some other method, and need to be passed through to drizzle 'as is', otherwise those updates will be over-written by this update.

Note: This parameter was preserved in the API for compatibility purposes with existing user processing pipe-lines. However, it has been removed from the `TEAL` interface because it is easy to have it set to `'Yes'` (especially between consecutive runs of `AstroDrizzle`) with potentially disastrous effects on input image `WCS` (for example it could wipe-out previously aligned `WCS`).

See also:

drizzlepac.adrizzle
Apply the 'drizzle' algorithm to the images

drizzlepac.ablot
Apply the 'blot' algorithm to drizzled images

drizzlepac.sky
: Perform sky subtraction

stsci.skypac.skymatch
: Sky computation and equalization

drizzlepac.createMedian
: Create a median combined image from a set of drizzled images

drizzlepac.drizCR
: Identify cosmic-rays by comparing blotted, median images to the original input images.

Notes

Something to keep in mind is that the full `AstroDrizzle` interface will make backup copies of your original files and place them in the `'OrIg_files'` directory of you current working directory.

All calibrated input images must have been updated using `updatewcs` from the `STWCS` package, to include the full distortion model in the header. Alternatively, one can set `updatewcs` parameter to `True` when running either `TweakReg` or `AstroDrizzle` from command line (Python interpreter) **the first time** on such images.

Examples

The `AstroDrizzle` task can be run from either the `TEAL` GUI or from Python. These examples illustrate the various syntax options available.

Example 1: Drizzle a set of calibrated (`_flt.fits`) images using mostly default parameters. Select the desired 'World Coordinate System' (WCS) aligned/updated by `TweakReg`. Let's say this WCS is stored with *key* `'A'` (different from the primary WCS). When combining images, ignore pixel flags of 64 and 32 in the DQ array of the (`_flt.fits`) images. Align the final product such that North is up, and set the final pixel scale to 0.05 arcseconds/pixel.

Run the task from Python using the command line.

```
>>> import drizzlepac
>>> from drizzlepac import astrodrizzle
>>> astrodrizzle.AstroDrizzle('*flt.fits', output='final',
...     wcskey='A', driz_sep_bits='64,32', final_wcs=True,
...     final_scale=0.05, final_rot=0)

Or, run the same task from the Python command line, but specify all
parameters in a config file named ``myparam.cfg``:

>>> astrodrizzle.AstroDrizzle('*flt.fits', configobj='myparam.cfg')
```

Help can be accessed via the "Help" pulldown menu in the `TEAL` GUI.
: It can also be accessed from the Python command-line and saved to a text file:

```
>>> from drizzlepac import astrodrizzle
>>> astrodrizzle.help()
```

or

```
>>> astrodrizzle.help(file='help.txt')
```

drizzlepac.astrodrizzle.**help**(*file=None*)

Print out syntax help for running astrodrizzle.

Parameters

file

[str (Default = None)] If given, write out help to the filename specified by this parameter Any previously existing file with this name will be deleted before writing out the help.

2.2 Process Input (data validation)

This module supports the interpretation and initial verification of all the input files specified by the user. These functions:

- reads in parameter values from MDRIZTAB reference file and merges those merges those values in with the rest of the parameters from the GUI/configObj, if use of MDRIZTAB was specified
- insure that all input files are multi-extension FITS files and converts them if they are not
- updates all input WCS's to be consistent with IDCTAB, if specified
- generates the ImageObject instances for each input file
- resets the DQ bits if specified by the user
- adds info about any user-provided IVM files to the ImageObjects
- generates the output WCS based on user inputs

Process input to MultiDrizzle/PyDrizzle.

Authors

Warren Hack

License

License

The input can be one of:

- a python list of files
- a comma separated string of filenames (including wild card characters)
- an association table
- an @file (can have a second column with names of ivm files)

No mixture of instruments is allowed. No mixture of association tables, @files and regular fits files is allowed. Files can be in GEIS or MEF format (but not waiver fits).

Runs some sanity checks on the input files. If necessary converts files to MEF format (this should not be left to `makewcs` because `updatewcs` may be `False`). Runs makewcs. The function `process_input` returns an association table, ivmlist, output name

The common interface interpreter for MultiDrizzle tasks, 'processCommonInput()', not only runs 'process_input()' but 'createImageObject()' and 'defineOutput()' as well to fully setup all inputs for use with the rest of the MultiDrizzle steps either as stand-alone tasks or internally to MultiDrizzle itself.

`drizzlepac.processInput.`**`addIVMInputs`**(*imageObjectList*, *ivmlist*)

Add IVM filenames provided by user to outputNames dictionary for each input imageObject.

`drizzlepac.processInput.`**`applyContextPar`**(*imageObjectList*, *contextpar*)

Apply the value of the parameter `context` to the input list, setting the name of the output context image to None if `context` is False

`drizzlepac.processInput.`**`buildASNList`**(*rootnames*, *asnname*, *check_for_duplicates=True*)

Return the list of filenames for a given set of rootnames

`drizzlepac.processInput.`**`buildEmptyDRZ`**(*input*, *output*)

Create an empty DRZ file.

This module creates an empty DRZ file in a valid FITS format so that the HST pipeline can handle the Multidrizzle zero expossure time exception where all data has been excluded from processing.

Parameters

input
[str] filename of the initial input to process_input

output
[str] filename of the default empty _drz.fits file to be generated

Notes

The existance of the DRZ file is also necessary to satisfy the interface expectations of the archive. Do NOT delete this file.

`drizzlepac.processInput.`**`buildFileList`**(*input*, *output=None*, *ivmlist=None*, *wcskey=None*, *updatewcs=True*, ***workinplace*)

Builds a file list which has undergone various instrument-specific checks for input to MultiDrizzle, including splitting STIS associations.

`drizzlepac.processInput.`**`buildFileListOrig`**(*input*, *output=None*, *ivmlist=None*, *wcskey=None*, *updatewcs=True*, ***workinplace*)

Builds a file list which has undergone various instrument-specific checks for input to MultiDrizzle, including splitting STIS associations. Compared to buildFileList, this version returns the list of the original file names as specified by the user (e.g., before GEIS->MEF, or WAIVER FITS->MEF conversion).

drizzlepac.processInput.**changeSuffixinASN**(*asnfile*, *suffix*)

Create a copy of the original asn file and change the name of all members to include the suffix.

drizzlepac.processInput.**checkDGEOFile**(*filenames*)

Verify that input file has been updated with NPOLFILE

This function checks for the presence of 'NPOLFILE' kw in the primary header when 'DGEOFILE' kw is present and valid (i.e. 'DGEOFILE' is not blank or 'N/A'). It handles the case of science files downloaded from the archive before the new software was installed there. If 'DGEOFILE' is present and 'NPOLFILE' is missing, print a message and let the user choose whether to (q)uit and update the headers or (c)ontinue and run astrodrizzle without the non-polynomial correction. 'NPOLFILE' will be populated in the pipeline before astrodrizzle is run.

In the case of WFPC2 the old style dgeo files are used to create detector to image correction at runtime.

Parameters

filenames
[list of str] file names of all images to be checked

drizzlepac.processInput.**checkForDuplicateInputs**(*rootnames*)

Check input files specified in ASN table for duplicate versions with multiple valid suffixes (_flt and _flc, for example).

drizzlepac.processInput.**checkMultipleFiles**(*input*)

Evaluates the input to determine whether there is 1 or more than 1 valid input file.

drizzlepac.processInput.**createImageObjectList**(*files*, *instrpars*, *output=None*, *group=None*, *undistort=True*, *inmemory=False*)

Returns a list of imageObject instances, 1 for each input image in the list of input filenames.

drizzlepac.processInput.**getMdriztabPars**(*input*)

High-level function for getting the parameters from MDRIZTAB

Used primarily for TEAL interface.

drizzlepac.processInput.**manageInputCopies**(*filelist*, ***workinplace*)

Creates copies of all input images in a sub-directory.

The copies are made prior to any processing being done to the images at all, including updating the WCS keywords. If there are already copies present, they will NOT be overwritten, but instead will be used to over-write the current working copies.

drizzlepac.processInput.**processFilenames**(*input=None*, *output=None*, *infilesOnly=False*)

Process the input string which contains the input file information and return a filelist,output

drizzlepac.processInput.**process_input**(*input*, *output=None*, *ivmlist=None*, *updatewcs=True*, *prodonly=False*, *wcskey=None*, ***workinplace*)

Create the full input list of filenames after verifying and converting files as needed.

drizzlepac.processInput.**reportResourceUsage**(*imageObjectList*, *outwcs*, *num_cores*, *interactive=False*)

Provide some information to the user on the estimated resource usage (primarily memory) for this run.

drizzlepac.processInput.**resetDQBits**(*imageObjectList*, *cr_bits_value=4096*)

Reset the CR bit in each input image's DQ array

drizzlepac.processInput.**runmakewcs**(*input*)

Runs make wcs and recomputes the WCS keywords

Parameters

input
[str or list of str] a list of files

Returns

output
[list of str] returns a list of names of the modified files (For GEIS files returns the translated names.)

drizzlepac.processInput.**setCommonInput**(*configObj*, *createOutwcs=True*, *overwrite_dict={}*)

The common interface interpreter for MultiDrizzle tasks which not only runs 'process_input()' but 'createImageObject()' and 'defineOutput()' as well to fully setup all inputs for use with the rest of the MultiDrizzle steps either as stand-alone tasks or internally to MultiDrizzle itself.

Parameters

configObj
[object] configObj instance or simple dictionary of input parameters

createOutwcs
[bool] whether of not to update the output WCS infomation

overwrite_dict: dict
dictionary of user input paramters to overwrite in configObj (needed in particular for using mdriztab=True option)

Notes

At a minimum, the configObj instance (dictionary) should contain:
configObj = { 'input':None,'output':None }

If provided, the configObj should contain the values of all the multidrizzle parameters as set by the user with TEAL. If no configObj is given, it will retrieve the default values automatically. In either case, the values from the input_dict will be merged in with the configObj before being used by the rest of the code.

Examples

You can set *createOutwcs=False* for the cases where you only want the images processed and no output wcs information in necessary; as in:

```
>>> imageObjectList,outwcs = processInput.processCommonInput(configObj)
```

drizzlepac.processInput.**update_member_names**(*oldasndict*, *pydr_input*)

Update names in a member dictionary.

Given an association dictionary with rootnames and a list of full file names, it will update the names in the member dictionary to contain '_*' extension. For example a rootname of 'u9600201m' will be replaced by 'u9600201m_c0h' making sure that a MEf file is passed as an input and not the corresponding GEIS file.

drizzlepac.processInput.**userStop**(*message*)

2.3 Static Mask Step

This step focuses entirely on creating a static mask for each detector used in the input images of all negative (presumably bad) pixel values. This module provides functions and classes that manage the creation of the global static masks.

For `staticMask`, the user interface function is *createMask()*.

Authors

Ivo Busko, Christopher Hanley, Warren Hack, Megan Sosey

License

License

drizzlepac.staticMask.**buildSignatureKey**(*signature*)

Build static file filename suffix used by mkstemp()

drizzlepac.staticMask.**constructFilename**(*signature*)

Construct an output filename for the given signature:

```
signature=[instr+detector,(nx,ny),detnum]
```

The signature is in the image object.

drizzlepac.staticMask.**createMask**(*input=None*, *static_sig=4.0*, *group=None*, *editpars=False*, *configObj=None*, ***inputDict*)

```
createMask(input=None, static_sig=4.0, group=None, editpars=False, \
configObj=None, **inputDict)
```

Create a static mask for all input images. The mask contains pixels that fall more than `static_sig` RMS below the mode for a given chip or extension. Those severely negative, or low pixels, might result from oversubtraction of bad pixels in the dark image, or high sky levels during calibration. For

example, each ACS WFC image contains a separate image for each of 2 CCDs, and seperate masks will be generated for each chip accordingly.

The final static mask for each chip contains all of the bad pixels that meet this criteria from all of the input images along with any bad pixels that satisfy the final_bits value specified by the user, and found in the images DQ mask.

Users should consider the details of their science image and decide whether or not creating this mask is appropriate for their resulting science. For example, if your field is very crowded, or contains mostly nebulous or extended objects, then the statistcs could be heavily skewed and the mask could end up containing sources.

The generated static masks are saved to disk for use in later steps with the following naming convention:

[Instrument][Detector]_[xsize]x[ysize]_[detector number]_staticMask.fits

so an ACS image would produce a static mask with the name:

ACSWFC_2048x4096_1_staticMask.fits

and this would be the only file saved to disk, storing the logic and of all the badpixel masks created for each acs image in the set.

For more information on the science applications of the static mask task, see the DrizzlePac Handbook

Parameters

input
[str, None (Default = None)] A list of images or associations you would like to use to compute the mask.

static
[bool (Default = True)] Create a static bad-pixel mask from the data? This mask flags all pixels that deviate by more than a value of `static_sig` sigma below the image median, since these pixels are typically the result of bad pixel oversubtraction in the dark image during calibration.

static_sig
[float (Default = 4.0)] The number of sigma below the RMS to use as the clipping limit for creating the static mask.

editpars
[bool (Default = False)] Set to `True` if you would like to edit the parameters using the GUI interface.

Examples

These tasks are designed to work together seemlessly when run in the full `AstroDrizzle` interface. More advanced users may wish to create specialized scripts for their own datasets, making use of only a subset of the predefined `AstroDrizzle` tasks, or add additional processing, which may be usefull for their particular data. In these cases, individual access to the tasks is important.

Something to keep in mind is that the full `AstroDrizzle` interface will make backup copies of your original files and place them in the `OrIg/` directory of your current working directory. If you are working with the stand alone interfaces, it is assumed that the user has already taken care of backing up their original datafiles as the input file with be directly altered.

Basic example of how to call static yourself from a python command line, using the default parameters for the task.

```
>>> from drizzlepac import staticMask
>>> staticMask.createMask('*flt.fits')
```

drizzlepac.staticMask.**createStaticMask**(*imageObjectList=[]*, *configObj=None*, *procSteps=None*)

drizzlepac.staticMask.**run**(*configObj*)

class drizzlepac.staticMask.**staticMask**(*configObj=None*)

This class manages the creation of the global static mask which masks pixels that are unwanted in the SCI array. A static mask object gets created for each global mask needed, one for each chip from each instrument/detector. Each static mask array has type Int16, and resides in memory.

Notes

Class that manages the creation of a global static mask which is used to mask pixels that are some sigma BELOW the mode computed for the image.

addMember(*imagePtr=None*)

Combines the input image with the static mask that has the same signature.

Parameters

imagePtr
[object] An imageObject reference

Notes

The signature parameter consists of the tuple:

```
(instrument/detector, (nx,ny), chip_id)
```

The signature is defined in the image object for each chip

close()

Deletes all static mask objects.

deleteMask(*signature*)

Delete just the mask that matches the signature given.

getFilename(*signature*)

Returns the name of the output mask file that should reside on disk for the given signature.

getMaskArray(*signature*)

Returns the appropriate StaticMask array for the image.

getMaskname(*chipid*)

Construct an output filename for the given signature:

```
signature=[instr+detector,(nx,ny),detnum]
```

The signature is in the image object and the name of the static mask file is saved as sci_chip.outputNames["staticMask"].

saveToFile(*imageObjectList*)

Saves the static mask to a file it uses the signatures associated with each mask to contruct the filename for the output mask image.

drizzlepac.staticMask.**help**(*file=None*)

Print out syntax help for running staticMask.

Parameters

file

[str (Default = None)] If given, write out help to the filename specified by this parameter Any previously existing file with this name will be deleted before writing out the help.

2.4 Sky-Subtraction Step

This step measures, subtracts and/or equalizes the sky from each input image while recording the subtracted value in the image header.

Authors

Christopher Hanley, Megan Sosey, Mihai Cara

License

License

drizzlepac.sky.**help**(*file=None*)

Print out syntax help for running sky.

Parameters

file

[str (Default = None)] If given, write out help to the filename specified by this parameter Any previously existing file with this name will be deleted before writing out the help.

drizzlepac.sky.**sky**(*input=None*, *outExt=None*, *configObj=None*, *group=None*, *editpars=False*, ***inputDict*)

Function for computing and subtracting (or equalizing/matching) the backgroud in input images. The algorithm for sky subtraction can be selected through the `skymethod` parameter. This function will update the `MDRIZSKY` keyword in the headers of the input files.

Sky subtraction is generally recommended for optimal flagging and removal of CR's when the sky background is more than a few electrons. However, some science applications may require the sky to not be removed, allowing for the final drizzle step to be performed with no sky subtraction. If you turn off sky subtraction, you should also set drizzle.pixfrac to 1, otherwise variations in sky between images will add noise to your data.

In addition to the "pure" sky computation, this task can be used for sky "equalization", that is, it can match sky values in the images that are part of a mosaic.

For cameras with multiple detectors (such as ACS/WFC, WFPC2, or WFC3), the sky values in each exposure are first measured separately for the different detectors. These different values are then compared, and the lowest measured sky value is used as the estimate for all of the detectors for that exposure. This is based on the premise that for large extended or bright targets, the pixel intensity distribution in one or more of the detectors may be significantly skewed toward the bright end by the target itself, thereby overestimating the sky on that detector. If the other detector is less affected by such a target, then its sky value will be lower, and can therefore also be substituted as the sky value for the detector with the bright source.

For more information on the science applications of the sky task, see the DrizzlePac Handbook:.

Parameters

input
[str or list of str (Default = None)] A Python list of image filenames, or just a single filename.

outExt
[str (Default = None)] The extension of the output image. If the output already exists then the input image is overwritten.

configObj
[configObject (Default = None)] An instance of `configObject`

group
[int (Default = None)] The group of the input image.

editpars
[bool (Default = False)] A parameter that allows user to edit input parameters by hand in the GUI.

inputDict
[dict, optional] An optional list of parameters specified by the user.

Note: These are parameters that `configObj` should contain by default. These parameters can be altered on the fly using the `inputDict`. If `configObj` is set to None and there is no `inputDict` information, then the values for the parameters

will be pulled from the default configuration files for the task. These are the same configuration files that are referenced in the TEAL parameter interface GUI. If you wish to edit these parameters by hand in the GUI, simply set editpars=True.

Table of optional parameters that should be in `configobj` and can also be specified in `inputDict`.

Name	Definition
`skyuser`	'KEYWORD in header which indicates a sky subtraction value to use'.
`skymethod`	'Sky computation method'
`skysub`	'Perform sky subtraction?'
`skywidth`	'Bin width of histogram for sampling sky statistics (in sigma)'
`skystat`	'Sky correction statistics parameter'
`skylower`	'Lower limit of usable data for sky (always in electrons)'
`skyupper`	'Upper limit of usable data for sky (always in electrons)'
`skyclip`	'Number of clipping iterations'
`skylsigma`	'Lower side clipping factor (in sigma)'
`skyusigma`	'Upper side clipping factor (in sigma)'
`skymask_cat`	'Catalog file listing image masks'
`use_static`	'Use static mask for skymatch computations?'
`sky_bits`	'Bit flags for identifying bad pixels in DQ array'
`skyuser`	'KEYWORD indicating a sky subtraction value if done by user'
`skyfile`	'Name of file with user-computed sky values'
`in_memory`	'Optimize for speed or for memory use'

These optional parameters are described in more detail below in the "Other Parameters" section.

Returns

None
[The input file's primary headers is updated with the computed sky value.]

Other Parameters

skysub
[bool (Default = Yes)] Turn on or off sky subtraction on the input data. When `skysub` is set to `no`, then `skyuser` field will be enabled and if user specifies a header keyword showing the sky value in the image, then that value will be used for CR-rejection but it will not be subtracted from the (drizzled) image data. If user sets `skysub` to `yes` then `skyuser` field will be disabled (and if it is not empty - it will be ignored) and user can use one of the methods available through the `skymethod` parameter to compute the sky or provide a file (see `skyfile` parameter) with values that should be subtracted from (single) drizzled images.

skymethod
[{'localmin', 'globalmin+match', 'globalmin', 'match'}, optional (Default = 'lo-

calmin')] Select the algorithm for sky computation:

- **'localmin'**: compute a common sky for all members of *an exposure* (see NOTES below). For a typical use, it will compute sky values for each chip/image extension (marked for sky subtraction in the `input` parameter) in an input image, and it will subtract the previously found minimum sky value from all chips (marked for sky subtraction) in that image. This process is repeated for each input image.

  Note: This setting is recommended when regions of overlap between images are dominated by "pure" sky (as opposite to extended, diffuse sources).

  Note: This is similar to the "skysub" algorithm used in previous versions of astrodrizzle.

- **'globalmin'**: compute a common sky value for all members of *all exposures* (see NOTES below). It will compute sky values for each chip/image extension (marked for sky subtraction in the `input` parameter) in **all** input images, find the minimum sky value, and then it will subtract the **same** minimum sky value from **all** chips (marked for sky subtraction) in **all** images. This method *may* useful when input images already have matched background values.
- **'match'**: compute differences in sky values between images in common (pairwise) sky regions. In this case computed sky values will be relative (delta) to the sky computed in one of the input images whose sky value will be set to (reported to be) 0. This setting will "equalize" sky values between the images in large mosaics. However, this method is not recommended when used in conjunction with AstroDrizzle because it computes relative sky values while `AstroDrizzle` needs "measured" sky values for median image generation and CR rejection.
- **'globalmin+match'**: first find a minimum "global" sky value in all input images and then use **'match'** method to equalize sky values between images.

  Note: This is the *recommended* setting for images containing diffuse sources (e.g., galaxies, nebulae) covering significant parts of the image.

  ---

skywidth
: [float, optional (Default Value = 0.1)] Bin width, in sigma, used to sample the distribution of pixel flux values in order to compute the sky background statistics.

skystat
: [{'median', 'mode', 'mean'}, optional (Default Value = 'median')] Statistical method for determining the sky value from the image pixel values.

skylower
: [float, optional (Default Value = INDEF)] Lower limit of usable pixel values for computing the sky. This value should be specified in the units of the input image.

skyupper
[float, optional (Default Value = INDEF)] Upper limit of usable pixel values for computing the sky. This value should be specified in the units of the input image.

skyclip
[int, optional (Default Value = 5)] Number of clipping iterations to use when computing the sky value.

skylsigma
[float, optional (Default Value = 4.0)] Lower clipping limit, in sigma, used when computing the sky value.

skyusigma
[float, optional (Default Value = 4.0)] Upper clipping limit, in sigma, used when computing the sky value.

skymask_cat
[str, optional (Default Value = '')] File name of a catalog file listing user masks to be used with images.

use_static
[bool, optional (Default Value = True)] Specifies whether or not to use static mask to exclude masked image pixels from sky computations.

sky_bits
[int, None, optional (Default = 0)] Integer sum of all the DQ bit values from the input image's DQ array that should be considered "good" when building masks for sky computations. For example, if pixels in the DQ array can be combinations of 1, 2, 4, and 8 flags and one wants to consider DQ "defects" having flags 2 and 4 as being acceptable for sky computations, then `sky_bits` should be set to 2+4=6. Then a DQ pixel having values 2,4, or 6 will be considered a good pixel, while a DQ pixel with a value, e.g., 1+2=3, 4+8=12, etc. will be flagged as a "bad" pixel.

Default value (0) will make *all* non-zero pixels in the DQ mask to be considered "bad" pixels, and the corresponding image pixels will not be used for sky computations.

Set `sky_bits` to `None` to turn off the use of image's DQ array for sky computations.

Note: DQ masks (if used), *will be* combined with user masks specified in the input @-file.

---

skyfile
[str, optional (Default Value = '')] Name of file containing user-computed sky values to be used with each input image. This ASCII file should only contain 2

columns: image filename in column 1 and sky value in column 2. The sky value should be provided in units that match the units of the input image and for multi-chip images, the same value will be applied to all chips.

skyuser
[str (Default = '')] Name of header keyword which records the sky value already subtracted from the image by the user. The `skyuser` parameter is ignored when `skysub` is set to `yes`.

Note: When `skysub``=``no` and `skyuser` field is empty, then `AstroDrizzle` will assume that sky background is 0.0 for the purpose of cosmic-ray rejection.

---

in_memory
[bool, optional (Default Value = False)] Specifies whether to optimize execution for speed (maximum memory usage) or use a balanced approach in which a minimal amount of image data is kept in memory and retrieved from disk as needed. The default setting is recommended for most systems.

Notes

sky() provides new algorithms for sky value computations and enhances previously available algorithms used by, e.g., Astrodrizzle.

First, the standard sky computation algorithm (see `skymethod = 'localmin'`) was upgraded to be able to use DQ flags and user supplied masks to remove "bad" pixels from being used for sky statistics computations.

Second, two new methods have been introduced: `'globalmin'` and `'match'`, as well as a combination of the two – `'globalmin+match'`.

- The `'globalmin'` method computes the minimum sky value across *all* chips in *all* input images. That sky value is then considered to be the background in all input images.
- The `'match'` algorithm is somewhat similar to the traditional sky subtraction method (`skymethod='localmin'`) in the sense that it measures the sky indipendently in input images (or detector chips). The major differences are that, unlike the traditional method,
  1. `'match'` algorithm computes *relative* sky values with regard to the sky in a reference image chosen from the input list of images; *and*
  2. Sky statistics is computed only in the part of the image that intersects other images.

  This makes `'match'` sky computation algorithm particularly useful for "equalizing" sky values in large mosaics in which one may have only (at least) pair-wise intersection of images without having a common intersection region (on the sky) in all images.

  The `'match'` method works in the following way: for each pair of intersecting images, an equation is written that requires that average surface brightness in the overlapping part of the sky be equal in both images. The final system of equations is then solved for unknown background levels.

> **Warning:** Current algorithm is not capable of detecting cases when some groups of intersecting images (from the input list of images) do not intersect at all other groups of intersecting images (except for the simple case when *single* images do not intersect any other images). In these cases the algorithm will find equalizing sky values for each group. However since these groups of images do not intersect each other, sky will be matched only within each group and the "inter-group" sky mismatch could be significant.
>
> Users are responsible for detecting such cases and adjusting processing accordingly.

> **Warning:** Because this method computes *relative sky values* compared to a reference image (which will have its sky value set to 0), the sky values computed with this method usually are smaller than the "absolute" sky values computed, e.g., with the `'localmin'` algorithm. Since AstroDrizzle expects "true" (as opposite to *relative*) sky values in order to correctly compute the median image or to perform cosmic-ray detection, this algorithm in not recommended to be used *alone* for sky computations to be used with `AstroDrizzle`.
>
> For the same reason, IVM weighting in `AstroDrizzle` should **not** be used with `'match'` method: sky values reported in `MDRIZSKY` header keyword will be relative sky values (sky offsets) and derived weights will be incorrect.

- The `'globalmin+match'` algorithm combines `'match'` and `'globalmin'` methods in order to overcome the limitation of the `'match'` method described in the note above: it uses `'globalmin'` algorithm to find a baseline sky value common to all input images and the `'match'` algorithm to "equalize" sky values in the mosaic. Thus, the sky value of the "reference" image will be equal to the baseline sky value (instead of 0 in `'match'` algorithm alone) making this method acceptable for use in conjunction with `AstroDrizzle`.

Glossary:

Exposure – a *subset* of FITS image extensions in an input image that correspond to different chips in the detector used to acquire the image. The subset of image extensions that form an exposure is defined by specifying extensions to be used with input images (see parameter `input`).

See help for `parse_at_line()` for details on how to specify image extensions.

Footprint – the outline (edge) of the projection of a chip or of an exposure on the celestial sphere.

Note:

- Footprints are managed by the spherical_geometry.polygon.SphericalPolygon class.
- Both footprints *and* associated exposures (image data, WCS information, and other header information) are managed by the `SkyLine` class.
- Each `SkyLine` object contains one or more `SkyLineMember` objects that manage both footprints *and* associated *chip* data that form an exposure.

Remarks:

- *sky()* works directly on *geometrically distorted* flat-fielded images thus avoiding the need to perform an additional drizzle step to perform distortion correction of input images.

  Initially, the footprint of a chip in an image is aproximated by a 2D planar rectangle representing the borders of chip's distorted image. After applying distortion model to this rectangle and progecting it onto the celestial sphere, it is approximated by spherical polygons. Footprints of exposures and mosaics are computed as unions of such spherical polygons while overlaps of image pairs are found by intersecting these spherical polygons.

Limitations and Discussions:

Primary reason for introducing "sky match" algorithm was to try to equalize the sky in large mosaics in which computation of the "absolute" sky is difficult due to the presence of large diffuse sources in the image. As discussed above, *sky()* accomplishes this by comparing "sky values" in a pair of images in the overlap region (that is common to both images). Quite obviously the quality of sky "matching" will depend on how well these "sky values" can be estimated. We use quotation marks around *sky values* because for some image "true" background may not be present at all and the measured sky may be the surface brightness of large galaxy, nebula, etc.

Here is a brief list of possible limitations/factors that can affect the outcome of the matching (sky subtraction in general) algorithm:

- Since sky subtraction is performed on *flat-fielded* but *not distortion corrected* images, it is important to keep in mind that flat-fielding is performed to obtain uniform surface brightness and not flux. This distinction is important for images that have not been distortion corrected. As a consequence, it is advisable that point-like sources be masked through the user-supplied mask files. Alternatively, one can use `upper` parameter to limit the use of bright objects in sky computations.
- Normally, distorted flat-fielded images contain cosmic rays. This algorithm does not perform CR cleaning. A possible way of minimizing the effect of the cosmic rays on sky computations is to use clipping (`nclip` > 0) and/or set `upper` parameter to a value larger than most of the sky background (or extended source) but lower than the values of most CR pixels.
- In general, clipping is a good way of eliminating "bad" pixels: pixels affected by CR, hot/dead pixels, etc. However, for images with complicated backgrounds (extended galaxies, nebulae, etc.), affected by CR and noise, clipping process may mask different pixels in different images. If variations in the background are too strong, clipping may converge to different sky values in different images even when factoring in the "true" difference in the sky background between the two images.
- In general images can have different "true" background values (we could measure it if images were not affected by large diffuse sources). However, arguments such as `lower` and `upper` will apply to all images regardless of the intrinsic differences in sky levels.

How to use the tasks stand alone interface in your own scripts:

These tasks are designed to work together seemlessly when run in the full `AstroDrizzle` interface. More advanced users may wish to create specialized scripts for their own datasets, making use of only a subset of the predefined `AstroDrizzle` tasks, or add additional processing, which may be usefull for their particular data. In these cases, individual access to the tasks is important.

Something to keep in mind is that the full `AstroDrizzle` interface will make backup copies of your original files and place them in the `OrIg/` directory of your current working directory. If you are

working with the stand alone interfaces, it is assumed that the user has already taken care of backing up their original datafiles as the input file with be directly altered.

Examples

Basic example of how to call sky yourself from a Python command line, this example will use the default parameter settings and subtract a sky value from each `*flt.fits` image in the current directory, saving the output file with the extension of "mysky":

```
>>> from drizzlepac import sky
>>> sky.sky('*flt.fits',outExt='mysky')
```

2.5 Image Drizzling Step

The operation of drizzling each input image needs to be performed twice during processing:

- single drizzle step: this initial step drizzles each image onto the final output WCS as separate images
- final drizzle step: this step produces the final combined image based on the cosmic-ray masks determined by *AstroDrizzle*

Interfaces to main drizzle functions.

Authors
Warren Hack

License
License

drizzlepac.adrizzle.**buildDrizParamDict**(*configObj*, *single=True*)

drizzlepac.adrizzle.**create_output**(*filename*, *arr*)

drizzlepac.adrizzle.**do_driz**(*insci*, *input_wcs*, *inwht*, *output_wcs*, *outsci*, *outwht*, *outcon*, *expin*, *in_units*, *wt_scl*, *wcslin_pscale=1.0*, *uniqid=1*, *pixfrac=1.0*, *kernel='square'*, *fillval='INDEF'*, *stepsize=10*, *wcsmap=None*)

Core routine for performing 'drizzle' operation on a single input image All input values will be Python objects such as ndarrays, instead of filenames. File handling (input and output) will be performed by calling routine.

drizzlepac.adrizzle.**drizFinal**(*imageObjectList*, *output_wcs*, *configObj*, *build=None*, *wcsmap=None*, *logfile=None*, *procSteps=None*)

drizzlepac.adrizzle.**drizSeparate**(*imageObjectList*, *output_wcs*, *configObj*, *logfile=None*, *wcsmap=None*, *procSteps=None*)

drizzlepac.adrizzle.**drizzle**(*input*, *outdata*, *wcsmap=None*, *editpars=False*, *configObj=None*, ***input_dict*)

Each input image gets drizzled onto a separate copy of the output frame. When stacked, these copies would correspond to the final combined product. As separate images, they allow for treatment of each input image separately in the undistorted, final WCS system. These images provide the information necessary for refining image registration for each of the input images. They also provide the images that will be succeedingly combined into a median image and then used for the subsequent blot and cosmic ray detection steps.

Aside from the input parameters, this step requires:

- valid input images with SCI extensions
- valid distortion coefficients tables
- any optional secondary distortion correction images
- numpy object (in memory) for static mask

This step produces:

- singly drizzled science image (simple FITS format)
- singly drizzled weight images (simple FITS format)

These images all have the same WCS based on the original input parameters and those provided for this step; specifically, output shape, pixel size, and orientation, if any have been specified at all.

Other Parameters

driz_separate
: [bool (Default = No)] This parameter specifies whether or not to drizzle each input image onto separate output images. The separate output images will all have the same WCS as the final combined output frame. These images are used to create the median image, needed for cosmic ray rejection.

driz_sep_kernel
: [{'square', 'point', 'turbo', 'gaussian', 'lanczos3'} (Default = 'turbo')] Used for the initial separate drizzling operation only, this parameter specifies the form of the kernel function used to distribute flux onto the separate output images. The current options are:

  - **square**: original classic drizzling kernel
  - **point**: this kernel is a point so each input pixel can only contribute to the single pixel that is closest to the output position. It is equivalent to the limit as `pixfrac` -> 0, and is very fast.
  - **gaussian**: this kernel is a circular gaussian with a FWHM equal to the value of `pixfrac`, measured in input pixels.
  - **turbo**: this is similar to kernel="square" but the box is always the same shape and size on the output grid, and is always aligned with the X and Y axes. This may result in a significant speed increase.
  - **lanczos3**: a Lanczos style kernel, extending a radius of 3 pixels from the center of the detection. The Lanczos kernel is a damped and bounded form of

the "sinc" interpolator, and is very effective for resampling single images when `scale=pixfrac=1`. It leads to less resolution loss than other kernels, and typically results in reduced correlated noise in outputs.

> **Warning:** While the `'gaussian'` and `'lanczos3'` kernels may produce reasonable results, we cannot guarantee that they will properly conserve flux; understand the effects of these kernels before using them.

> **Warning:** The `'lanczos3'` kernel tends to result in much slower processing as compared to other kernel options. This option should never be used for `pixfrac`!=1.0, and is not recommended for `scale` != 1.0.

The default for this step is **"turbo"** since it is much faster than **"square"**, and it is quite satisfactory for the purposes of generating the median image. More information about the different kernels can be found in the help file for the drizzle task.

driz_sep_wt_scl
[float (Default = exptime)] This parameter specifies the weighting factor for input image. If `driz_sep_wt_scl=exptime`, then the scaling value will be set equal to the exposure time found in the image header. The use of the default value is recommended for producing optimal behavior for most scenarious. It is possible to set `wt_scl`='expsq' for weighting by the square of the exposure time, which is optimal for read-noise dominated images.

driz_sep_pixfrac
[float (Default = 1.0)] Fraction by which input pixels are "shrunk" before being drizzled onto the output image grid, given as a real number between 0 and 1. This specifies the size of the footprint, or "dropsize", of a pixel in units of the input pixel size. If `pixfrac` is set to less than 0.001, the kernel parameter will be reset to 'point' for more efficient processing. In the step of drizzling each input image onto a separate output image, the default value of 1.0 is best in order to ensure that each output drizzled image is fully populated with pixels from the input image. For more information, see the help for the `drizzle` task.

driz_sep_fillval
[int or INDEF (Default = INDEF)] Value to be assigned to output pixels that have zero weight, or that receive flux from any input pixels during drizzling. This parameter corresponds to the `fillval` parameter of the `drizzle` task. If the default of `INDEF` is used, and if the weight in both the input and output images for a given pixel are zero, then the output pixel will be set to the value it would have had if the input had a non-zero weight. Otherwise, if a numerical value is provided (e.g. 0), then these pixels will be set to that value.

driz_sep_bits
[int (Default = 0)] Integer sum of all the DQ bit values from the input image's DQ

array that should be considered 'good' when building the weighting mask. This can also be used to reset pixels to good if they had been flagged as cosmic rays during a previous run of `AstroDrizzle`, by adding the value 4096 for ACS and WFPC2 data. Please see the section on Selecting the `Bits` Parameter for a more detailed discussion.

driz_sep_wcs
[bool (Default = No)] Define custom WCS for seperate output images?

driz_sep_refimage
[str (Default = '')] Reference image from which a WCS solution can be obtained.

driz_sep_rot
[float (Default = INDEF)] Position Angle of output image's Y-axis relative to North. A value of 0.0 would orient the final output image to be North up. The default of `INDEF` specifies that the images will not be rotated, but will instead be drizzled in the default orientation for the camera with the x and y axes of the drizzled image corresponding approximately to the detector axes. This conserves disk space, as these single drizzled images are only used in the intermediate step of creating a median image.

driz_sep_scale
[float (Default = INDEF)] Linear size of the output pixels in arcseconds/pixel for each separate drizzled image (used in creating the median for cosmic ray rejection). The default value of `INDEF` specifies that the undistorted pixel scale for the first input image will be used as the pixel scale for all the output images.

driz_sep_outnx
[int (Default = INDEF)] Size, in pixels, of the X axis in the output images that each input will be drizzled onto. If no value is specified, the smallest size that can accommodate the full dithered field will be used.

driz_sep_outny
[int (Default = INDEF)] Size, in pixels, of the Y axis in the output images that each input will be drizzled onto. If no value is specified, the smallest size that can accommodate the full dithered field will be used.

driz_sep_ra
[float (Default = INDEF)] Right ascension (in decimal degrees) specifying the center of the output image. If this value is not designated, the center will automatically be calculated based on the distribution of image dither positions.

driz_sep_dec
[float (Default = INDEF)] Declination (in decimal degrees) specifying the center of the output image. If this value is not designated, the center will automatically be calculated based on the distribution of image dither positions.

Notes

These tasks are designed to work together seemlessly when run in the full `AstroDrizzle` interface. More advanced users may wish to create specialized scripts for their own datasets, making use of only a subset of the predefined `AstroDrizzle` tasks, or add additional processing, which may be usefull for their particular data. In these cases, individual access to the tasks is important.

Something to keep in mind is that the full `AstroDrizzle` interface will make backup copies of your original files and place them in the `OrIg/` directory of your current working directory. If you are working with the stand alone interfaces, it is assumed that the user has already taken care of backing up their original datafiles as the input file with be directly altered.

There are two user interface function for this task, one to allow you to create seperately drizzled images of each image in your list and the other to create one single output drizzled image, which is the combination of all of them:

```
def drizSeparate(imageObjectList,output_wcs,configObj,wcsmap=wcs_functions.
↪WCSMap)
def drizFinal(imageObjectList, output_wcs, configObj,build=None,wcsmap=wcs_
↪functions.WCSMap)
if configObj[single_step]['driz_separate']:
    drizSeparate(imgObjList,outwcs,configObj,wcsmap=wcsmap)
else:
    drizFinal(imgObjList,outwcs,configObj,wcsmap=wcsmap)
```

Examples

Basic example of how to call static yourself from a Python command line, using the default parameters for the task.

```
>>> from drizzlepac import adrizzle
```

drizzlepac.adrizzle.**get_data**(*filename*)

drizzlepac.adrizzle.**interpret_maskval**(*paramDict*)

Apply logic for interpreting final_maskval value...

drizzlepac.adrizzle.**mergeDQarray**(*maskname*, *dqarr*)

Merge static or CR mask with mask created from DQ array on-the-fly here.

drizzlepac.adrizzle.**run**(*configObj*, *wcsmap=None*)

Interface for running `wdrizzle` from TEAL or Python command-line.

This code performs all file `I/O` to set up the use of the drizzle code for a single exposure to replicate the functionality of the original `wdrizzle`.

drizzlepac.adrizzle.**run_driz**(*imageObjectList*, *output_wcs*, *paramDict*, *single*, *build*, *wcsmap=None*)

Perform drizzle operation on input to create output. The input parameters originally was a list of dictionaries, one for each input, that matches the primary parameters for an `IRAF drizzle` task.

This method would then loop over all the entries in the list and run `drizzle` for each entry.

Parameters required for input in paramDict:
build,single,units,wt_scl,pixfrac,kernel,fillval, rot,scale,xsh,ysh,blotnx,blotny,outnx,outny,data

`drizzlepac.adrizzle.`**`run_driz_chip`**(*img*, *chip*, *output_wcs*, *outwcs*, *template*, *paramDict*, *single*, *doWrite*, *build*, *_versions*, *_numctx*, *_nplanes*, *_numchips*, *_outsci*, *_outwht*, *_outctx*, *_hdrlist*, *wcsmap*)

Perform the drizzle operation on a single chip. This is separated out from `run_driz_img` so as to keep together the entirety of the code which is inside the loop over chips. See the `run_driz` code for more documentation.

`drizzlepac.adrizzle.`**`run_driz_img`**(*img*, *chiplist*, *output_wcs*, *outwcs*, *template*, *paramDict*, *single*, *num_in_prod*, *build*, *_versions*, *_numctx*, *_nplanes*, *chipIdxCopy*, *_outsci*, *_outwht*, *_outctx*, *_hdrlist*, *wcsmap*)

Perform the drizzle operation on a single image. This is separated out from *run_driz()* so as to keep together the entirety of the code which is inside the loop over images. See the *run_driz()* code for more documentation.

`drizzlepac.adrizzle.`**`updateInputDQArray`**(*dqfile*, *dq_extn*, *chip*, *crmaskname*, *cr_bits_value*)

2.6 Median Image Computation Step

A median image gets generated from the stack of undistorted single drizzle images. Create a median image from the singly drizzled images.

Authors
Warren Hack, Mihai Cara

License
License

`drizzlepac.createMedian.`**`createMedian`**(*imgObjList*, *configObj*, *procSteps=None*)

Top-level interface to createMedian step called from top-level AstroDrizzle.

This function parses the input parameters then calls the `_median()` function to median-combine the input images into a single image.

`drizzlepac.createMedian.`**`median`**(*input=None*, *configObj=None*, *editpars=False*, ***inputDict*)

The singly drizzled science images are combined to create a single median image. This median combination gets performed section-by-section from the input single drizzled images. Each section corresponds to a contiguous set of lines from each image taking up no more than 1Mb in memory, such that combining 10 input images would only require 10Mb for these sections. The goal of this step is to establish an estimate for what the fully cleaned image should look like in order to enable better bad pixel detection, in addition to improving the alignment of the image stack. Creating a median image from the aligned and undistorted input images allows for a statistical rejection of bad pixels.

The final median image serves as the only output from this step.

For more information on the science applications of the `createMedian` task, see the DrizzlePac Handbook.

Parameters

input
[str or list of str (Default = None)] A Python list of drizzled image filenames, or just a single filename.

configObj
[configObject (Default = None)] An instance of `configObject` which overrides default parameter settings.

editpars
[bool (Default = False)] A parameter that allows user to edit input parameters by hand in the GUI. True to use the GUI to edit parameters.

inputDict
[dict, optional] An optional list of parameters specified by the user, which can also be used to override the defaults.

Other Parameters

median
[bool (Default = No)] This parameter specifies whether or not to create a median image. This median image will be used as the comparison 'truth' image :in the cosmic ray rejection step.

median_newmasks
[bool (Default = Yes)] This parameter specifies whether or not new mask files will be created when the median image is created. These masks are generated from weight files previously produced by the "driz_separate" step, and contain all bad pixel information used to exclude pixels when calculating the median. Generally this step should be set to "Yes", unless for some reason, it is desirable to include bad pixel information when generating the median.

combine_maskpt
[float (Default = 0.7)] Percentage of weight image values, below which the are flagged.

combine_type
[str {'average', 'median', 'sum', 'minmed'} (Default = 'minmed')] This parameter defines the method that will be used to create the median image. The 'average', 'median', and 'sum' options set the calculation type when running 'numcombine', a numpy method for median-combining arrays to create the median image. The "minmed" option will produce an image that is generally the same as the median, except in cases where the median is significantly higher than the minimum good pixel value. In this case, "minmed" will choose the minimum value. The sigma thresholds for this decision are provided by the "combine_nsigma" parameter. However, as the "combine_nsigma" parameter does not adjust for the larger probability of a single "nsigma" event with a greater number of images, "minmed" will bias the comparison image low for a large number of images. The value of sigma is computed as $\sigma = \sqrt{(M + S + R^2)}$, where *M* is the median image data (in electrons), *S* is the value of the subtracted sky (in electrons), and *R* is the value of the readout noise (in electrons). "minmed" is highly recommended for three

images, and is good for four to six images, but should be avoided for ten or more images.

A value of 'median' is the recommended method for a large number of images, and works equally well as minmed down to approximately four images. However, the user should set the "combine_nhigh" parameter to a value of 1 when using "median" with four images, and consider raising this parameter's value for larger numbers of images. As a median averages the two inner values when the number of values being considered is even, the user may want to keep the total number of images minus "combine_nhigh" odd when using "median".

combine_nsigma
[float (Default = '4 3')] This parameter defines the sigmas used for accepting minimum values, rather than median values, when using the 'minmed' combination method. If two values are specified the first value will be used in the initial choice between median and minimum, while the second value will be used in the "growing" step to reject additional pixels around those identified in the first step. If only one value is specified, then it is used in both steps.

combine_nlow
[int (Default = 0)] This parameter sets the number of low value pixels to reject automatically during image combination.

combine_nhigh
[int (Default = 0)] This parameter sets the number of high value pixels to reject automatically during image combination.

combine_lthresh
[float (Default = INDEF)] Sets the lower threshold for clipping input pixel values during image combination. This value gets passed directly to 'imcombine' for use in creating the median image. If the parameter is set to "None", no thresholds will be imposed.

combine_hthresh
[float (Default = INDEF)] This parameter sets the upper threshold for clipping input pixel values during image combination. The value for this parameter is passed directly to 'imcombine' for use in creating the median image. If the parameter is set to "None", no thresholds will be imposed.

combine_grow
[int (Default = 1)] Width, in pixels, beyond the limit set by the rejection algorithm being used, for additional pixels to be rejected in an image. This parameter is used to set the 'grow' parameter in 'imcombine' for use in creating the median image **only when** `combine_type` is `'(i)minmed'`. When `combine_type` is anything other than `'(i)minmed'`, this parameter is ignored (set to 0).

combine_bufsize
[float (Default = None)] Size of buffer, in MB (MiB), to use when reading in each section of each input image. The default buffer size is 1MB. The larger the buffer size, the fewer times the code needs to open each input image and the more memory will be required to create the median image. A larger buffer can be helpful when

using compression, since slower copies need to be made of each set of rows from each input image instead of using memory-mapping.

Examples

For `createMedian`, the user interface function is `median`:

```
>>> from drizzlepac import createMedian
>>> createMedian.median('*flt.fits')
```

`drizzlepac.createMedian.`**`run`**(*configObj*)

2.7 Median Image Blotting Step

In this step the median image now gets blotted back to create median-cleaned images which can be compared directly with each input image to identify cosmic-rays.

Authors
Warren Hack

License
License

`drizzlepac.ablot.`**`blot`**(*data*, *outdata*, *configObj=None*, *wcsmap=wcs_functions.WCSMap*, \
editpars=False, ***input_dict*)

The median image is the combination of the WCS aligned input images that have already had the distortion model applied. Taking the median of the aligned images allows for a statistical rejection of bad pixels from the image stack. The resulting median image can then be input for the blot task with the goal of creating 'cleaned' versions of the input images at each of their respective dither locations. These "blotted" images can then be directly compared to the original distorted input images for detection of image artifacts (i.e. bad-pixels, hot pixels, and cosmic-rays) whose locations will be saved to the output badpixel masks.

Aside from the input parameters, this step only requires opening the single median image created from all the input images. A distorted version of the median image corresponding to each input 'chip' (extension) is written as output from this step as separate simple FITS images.

For more information on the science applications of the blot task, see the DrizzlePac Handbook

Parameters

data
[str] Input distortion-corrected (median or drizzled) image to be used as the source for creating blotted images.

reference
[str] Filename of image to read to define the blotted WCS; image with distortion to be matched by output blotted image.

outdata
[str] Filename for output blotted image.

coeffs
[bool (Default Value = True)] This parameters specifies whether or not to use the header-based distortion coefficients when creating the blotted, distorted image. If False, no distortion will be applied at all, effectively working as a cut-out operation.

interp
[str{'nearest', 'linear', 'poly3', 'poly5', 'sinc'} (Default = 'poly5')] This parameter defines the method of interpolation to be used when blotting drizzled images back to their original WCS solution. Valid options include:

- **nearest**: Nearest neighbor
- **linear**: Bilinear interpolation in x and y
- **poly3**: Third order interior polynomial in x and y
- **poly5**: Fifth order interior polynomial in x and y
- **sinc**: Sinc interpolation (accurate but slow)

The 'poly5' interpolation method has been chosen as the default because it is relatively fast and accurate.

If 'sinc' interpolation is selected, then the value of the parameter for `blot_sinscl will be used to specify the size of the sinc interpolation kernel.

sinscl
[float (Default Value = 1.0)] Size of the sinc interpolation kernel in pixels.

stepsize
[int (Default Value = 10)] Number of pixels for WCS interpolation. The distortion model will be sampled exactly and completely every `stepsize pixel with bilinear interpolation being used to compute the distortion for intermediate pixels. This optimization speeds up the computation significantly when stepsize >> 1 at the expense of interpolation errors for intermediate pixels.

addsky
[bool (Default Value = Yes)] Add back a sky value using the MDRIZSKY value from the header. If 'Yes' (True), the blot_skyval parameter is ignored.

skyval
[float (Default Value = 0.0)] This is a user-specified custom sky value to be added to the blot image. This is only used if blot_addsky is 'No' (False).

in_units
[str{'cps', 'counts'} (Default Value= 'cps')] Units of input (drizzled) image. Valid options are **'cps'** and **'counts'**.

out_units
[str{'cps', 'counts'} (Default Value = 'counts')] Units of the ouput (blotted) image. Valid options are **'cps'** and **'counts'**.

expkey
[str (Default Value = 'exptime)] Name of keyword to use to extract exposure time value, which will be used to scale the blotted image to the final output flux values when `out_units` is set to **counts**.

expout
[str or float (Default Value = 'input')] Value of exposure time to use in scaling the output blotted image when `out_units` is set to **counts**. If set to **'input'**, the value will be read in from the input image header keyword specified by `expkey`.

Note: The following parameters, when set, will override any value determined from `refimage` if a reference image was specified.

---

outscale
[float,optional] Absolute size of output pixels in arcsec/pixel

orient
[float] Orientation of output (PA of Y axis, N through E)

raref
[float] RA of reference point on output image(CRVAL1,degrees)

decref
[float] Dec of reference point on output image (CRVAL2, degrees)

xrefpix
[float] Reference pixel X position on output (CRPIX1)

yrefpix
[float] Reference pixel Y position on output (CRPIX2)

outnx
[float] Size of output image's X-axis (pixels)

outny
[float] Size of output image's Y-axis (pixels)

Notes

These tasks are designed to work together seemlessly when run in the full `AstroDrizzle` interface. More advanced users may wish to create specialized scripts for their own datasets, making use of only a subset of the predefined `AstroDrizzle` tasks, or add additional processing, which may be usefull for their particular data. In these cases, individual access to the tasks is important.

Something to keep in mind is that the full `AstroDrizzle` interface will make backup copies of your original files and place them in the `OrIg/` directory of your current working directory. If you are working with the stand alone interfaces, it is assumed that the user has already taken care of backing up their original datafiles as the input file with be directly altered.

Examples

1. Basic example of how to call *blot()* yourself from a python command line, using the default parameter settings:

```
>>> from drizzlepac import ablot
>>> ablot.blot()
```

2. Creation of a blotted image from the products generated by running the AstroDrizzle task can be done for the median image "adriz_aligned_wcs_f814w_med.fits" to re-create the (SCI,1) chip from "j8c0d1bwq_flc.fits" using:

```
>>> from drizzlepac import ablot
>>> from stsci.tools import teal
>>> blotobj = teal.load('ablot') # get default values
>>> ablot.blot('adriz_aligned_wcs_f814w_med.fits','j8c0d1bwq_flc.fits[sci,1]
↪',
'aligned_f814w_sci1_blot.fits',configObj=blotobj)
```

or

```
>>> a = teal.teal('ablot')
# set data = adriz_aligned_wcs_f814w_med.fits
# set reference = j8c0d1bwq_flc.fits[sci,1]
# set outdata = aligned_f814w_sci1_blot.fits
```

drizzlepac.ablot.**help**(*file=None*)

Print out syntax help for running ablot.

Parameters

file
[str (Default = None)] If given, write out help to the filename specified by this parameter Any previously existing file with this name will be deleted before writing out the help.

drizzlepac.ablot.**runBlot**(*imageObjectList*, *output_wcs*, *configObj={}*, *wcsmap=wcs_functions.WCSMap*, *procSteps=None*)

2.8 Cosmic-ray Identification Step

The cosmic rays and bad pixels are now identified by comparing the input images with the associated blotted,median-cleaned images created. Mask blemishes in dithered data by comparison of an image with a model image and the derivative of the model image.

Authors
Warren Hack

License

License

drizzlepac.drizCR.**createCorrFile**(*outfile*, *arrlist*, *template*)

Create a _cor file with the same format as the original input image.

The DQ array will be replaced with the mask array used to create the _cor file.

drizzlepac.drizCR.**drizCR**(*input=None*, *configObj=None*, *editpars=False*, ***inputDict*)

The blotted median images that are now transformed back into the original reference frame, get compared to the original input images to detect any spurious pixels (which may include pixels impacted by cosmic rays) in each input. Those spurious pixels then get flagged as 'bad' in the output cosmic ray mask files, which get used as input for the final combination so that they do not show up in the final product. The identified bad pixels get flagged by updating the input mask files. Optionally, copies of the original images with the bad pixels removed can be created through the use of the `driz_cr_corr` parameter.

Parameters

input
[str or list of str (Default = None)] A Python list of blotted median image filenames, or just a single filename.

configObj
[configObject (Default = None)] An instance of `configObject` which overrides default parameter settings.

editpars
[bool (Default = False)] A parameter that allows user to edit input parameters by hand in the GUI. True to use the GUI to edit parameters.

inputDict
[dict, optional] An optional list of parameters specified by the user, which can also be used to override the defaults.

Other Parameters

driz_cr
[bool (Default = False)] Perform cosmic-ray detection? If set to `True`, cosmic-rays will be detected and used to create cosmic-ray masks based on the algorithms from `deriv` and `driz_cr`.

driz_cr_corr
[bool (Default = False)] Create a cosmic-ray cleaned input image? I set to `True`, a cosmic-ray cleaned `_cor` image will be generated directly from the input image, and a corresponding `_crmask` file will be written to document detected pixels affected by cosmic-rays.

driz_cr_snr
[list of floats (Default = '3.5 3.0')] The values for this parameter specify the signal-to-noise ratios for the `driz_cr` task to be used in detecting cosmic rays. See the help file for `driz_cr` for further discussion of this parameter.

driz_cr_grow
[int (Default = 1)] The radius, in pixels, around each detected cosmic-ray, in which more stringent detection criteria for additional cosmic rays will be used.

driz_cr_ctegrow
[int (Default = 0)] Length, in pixels, of the CTE tail that should be masked in the drizzled output.

driz_cr_scale
[str (Default = '1.2 0.7')] Scaling factor applied to the derivative in `driz_cr` when detecting cosmic-rays. See the help file for `driz_cr` for further discussion of this parameter.

Notes

These tasks are designed to work together seemlessly when run in the full `AstroDrizzle` interface. More advanced users may wish to create specialized scripts for their own datasets, making use of only a subset of the predefined `AstroDrizzle` tasks, or add additional processing, which may be usefull for their particular data. In these cases, individual access to the tasks is important.

Something to keep in mind is that the full `AstroDrizzle` interface will make backup copies of your original files and place them in the `OrIg/` directory of your current working directory. If you are working with the stand alone interfaces, it is assumed that the user has already taken care of backing up their original datafiles as the input file with be directly altered.

Examples

Basic example of how to call `drizCR` yourself from a Python command line using the default parameters for the task.

```
>>> from drizzlepac import drizCR
>>> drizCR.drizCR('*flt.fits')
```

drizzlepac.drizCR.**run**(*configObj*)

drizzlepac.drizCR.**rundrizCR**(*imgObjList*, *configObj*, *procSteps=None*)

drizzlepac.drizCR.**setDefaults**(*configObj={}*)
Return a dictionary of the default parameters which also been updated with the user overrides.

drizzlepac.drizCR.**help**(*file=None*)
Print out syntax help for running `drizCR`.

Parameters

file
[str (Default = None)] If given, write out help to the filename specified by this parameter Any previously existing file with this name will be deleted before writing out the help.

2.9 Output Image Generation

This module manages the creation of the output image FITS file.

Authors
Warren Hack

License
License

class drizzlepac.outputimage.**OutputImage**(*plist*, *input_pars*, *build=True*, *wcs=None*, *single=False*, *blot=False*)

This class manages the creation of the array objects which will be used by Drizzle. The three arrays, SCI/WHT/CTX, will be setup either as extensions in a single multi-extension FITS file, or as separate FITS files.

The object 'plist' must contain at least the following members:

```
plist['output']   - name of output FITS image (for SCI)
plist['outnx']    - size of X axis for output array
plist['outny']    - size of Y axis for output array
```

If 'single=yes', then 'plist' also needs to contain:

```
plist['outsingle']
plist['outsweight']
plist['outscontext']
```

If 'blot=yes', then 'plist' also needs:

```
plist['data']
plist['blotImage']
plist['blotnx'],plist['blotny']
```

If 'build' is set to 'no', then each extension/array must be in separate FITS objects. This would also require:

```
plist['outdata']    - name of output SCI FITS image
plist['outweight']  - name of output WHT FITS image
plist['outcontext'] - name of output CTX FITS image
```

Optionally, the overall exposure time information can be passed as:

```
plist['texptime'] - total exptime for output
plist['expstart'] - start time of combined exposure
plist['expend']   - end time of combined exposure
```

addDrizKeywords(*hdr*, *versions*)

Add drizzle parameter keywords to header.

find_kwupdate_location(*hdr*, *keyword*)

Find the last keyword in the output header that comes before the new keyword in the original, full input headers. This will rely on the original ordering of keywords from the original input files in order to place the updated keyword in the correct location in case the keyword was removed from the output header prior to calling this method.

set_bunit(*bunit*)

Method used to update the value of the bunit attribute.

set_units(*units*)

Method used to record what units were specified by the user for the output product.

writeFITS(*template*, *sciarr*, *whtarr*, *ctxarr=None*, *versions=None*, *overwrite=True*, *blend=True*, *virtual=False*, *rules_file=None*, *logfile=None*)

Generate PyFITS objects for each output extension using the file given by 'template' for populating headers.

The arrays will have the size specified by 'shape'.

2.10 Index

- genindex
- modindex

CHAPTER

THREE

MAST DATA PRODUCTS

3.1 Standard Pipeline Data Products

3.1.1 Reproducing MAST DATA Products

`runastrodriz` is a module to control operation of astrodrizzle which removes distortion and combines HST images in the pipeline.

Typical Usage

```
>>> runastrodriz.py [-fhibn] inputFilename [newpath]
```

Alternative Usage

```
>>> python
>>> from wfc3tools import runastrodriz
>>> runastrodriz.process(inputFilename,force=False,newpath=None,inmemory=False)
```

GUI Usage under Python (Legacy only)

```
>>> python
>>> from stsci.tools import teal
>>> import wfc3tools
>>> cfg = teal.teal('runastrodriz')
```

Options

If the '-i' option gets specified, no intermediate products will be written out to disk. These products, instead, will be kept in memory. This includes all single drizzle products (*single_sci* and *single_wht*), median image, blot images, and crmask images. The use of this option will therefore require significantly more memory than usual to process the data.

If a value has been provided for the newpath parameter, all processing will be performed in that directory/ramdisk. The steps involved are:

- create a temporary directory under that directory named after the input file
- copy all files related to the input to that new directory
- change to that new directory and run astrodrizzle
- change back to original directory
- move (not copy) ALL files from temp directory to original directory
- delete temp sub-directory

The '-b' option will run this task in BASIC mode without creating headerlets for each input image.

The '-n' option allows the user to specify the number of cores to be used in running AstroDrizzle.

Note: This value will be forced to a value of '1' (one) on Windows systems due to exceptions caused by threaded logging under Windows. Future versions will lift this enforced restriction on Windows systems once issues with logging are resolved.

3.1.2 Pipeline Astrometric Calibration Description

A lot of effort goes into trying to determine the WCS solution which provides closest agreement with the GAIA astrometry frame. Combining the images successfully requires the most accurate alignment possible between all the input exposures taking into account the best available distortion model for the input exposures. Being able to compare the combined image to other exposures taken of the same field at other times, or even with other telescopes, requires that the WCS be defined to come as close to the GAIA frame as possible. The processing done by `drizzlepac.runastrodriz` attempts to not only apply the most current distortion model, but also the best available pre-computed GAIA-based WCS solutions while proceeding to determine it's own solution to the available GAIA reference stars for the field-of-view. It also performs numerous checks to see which of these solutions results in the most accurately aligned images to each other. The processing continued to attempt an alignment to GAIA using sources it identifies from the observations and checks to see if any fit was successful. Finally, it selects the WCS solution most closely aligned to GAIA that as the basis for creating the final, distorted-corrected, combined drizzle products for the set of exposures being processed.

Note: The API for the code used for these operations are described in more detail *HAP API*.

Overview

The overall logic implemented by `drizzlepac.runastrodriz` to generate the final set of drizzle products involves creating multiple sets of drizzle products and ultimately selecting one as the 'best'. The basic steps are outlined here, with following sections providing additional detail on each step.

1. Initialization

a) Get any defined environment variables to control processing

b) Interpret input file

c) Make sure all input files are in a local directory

d) Check for DRIZCORR calibration switch in input files

e) Create name for output log files

f) Define lists of individual input files to be processed

1. Run updatewcs without astrometry database update on all input exposures (FLCs? and FLTs)
2. Generate initial default products and perform verification
    a) perform cosmic-ray identification and generate drizzle products using astrodrizzle for all sets of inputs
    b) verify relative alignment with focus index after masking out CRs
    c) if alignment fails, update trailer file with failure information
3. If alignment is verified,
        a) copy inputs to separate sub-directory for processing
        b) run updatewcs to get a priori updates
        - apply 'best' apriori (not aposteriori) solution
        c) generate drizzle products for all sets of inputs (FLC and/or FLT)
        d) verify alignment using focus index on FLC or, if no FLC, FLT products
        e) if alignment fails, update trailer file with info on failure
        f) if product alignment verified:
            - copy all drizzle products to parent directory
            - copy updated input exposures to parent directory
4. If a posteriori correction enabled,
    a) copy all inputs to separate sub-directory for processing
    b) run align to align the images
    c) generate drizzle products for all sets of inputs (FLC and/or FLT) without CR identification

d) verify alignment using focus index on FLC or, if no FLC, FLT products

e) determine similarity index relative to pipeline default product

f) if either focus or similarity indicates a problem, update trailer file with info on failure

g) if product alignment verified:

- copy all drizzle products to parent directory
- copy updated input exposures to parent directory

5. Remove all processing sub-directories

Initialization

Environment Variables

The pipeline processing code starts out by looking to see whether the user has defined any processing behavior through the use of these environment variables:

- **'ASTROMETRY_COMPUTE_APOSTERIORI'**: This environment variable specifies whether or not to attempt an *a posteriori* alignment where the code looks for sources in each of the images and uses those positions to perform relative alignment between the images and then fit those images to the GAIA frame.
- **'ASTROMETRY_APPLY_APRIORI'**: This environment variable turns on/off application of any pre-defined(*a priori*) WCS solution found in the astrometry database.
- **'ASTROMETRY_STEP_CONTROL' [DEPRECATED, do not use]**: Old variable replaced by 'ASTROMETRY_APPLY_APRIORI'.

Values that can be provided for setting these variables are:

- 'on', 'yes', 'true': Any of these values will turn **on** the processing controlled by the variable
- 'off', 'no', 'false': Any of these values will turn **off** the processing controleed by the variable

By default, all the processing steps are turned **on** during pipeline processing in order to maximize the chances of aligning the data as closely as possible to the absolute astrometry standard coordinate system defined through the use of the GAIA catalogs. However, these controls are provided to support those observations which would not be suitable for such alignment, including observations of single sources.

Input Data

The processing code needs to be told what data to process, and for `drizzlepac.runastrodriz`, a single input filename is all that **can** be provided. This single input will be either:

- the name of an association table for a whole set of input exposures with a filename that looks like **'<rootname>_asn.fits'**, where <rootname> is the designation for the association, such as *'ie6d07030_asn.fits'*.
- the name of a single (uncalibrated) exposure with a filename that looks like **'<rootname>_raw.fits'**.

This one input filename, though, will simply provide the code with the information it needs to find all the calibrated input exposures which need to have their distortion-models updated and applied. The whole set of input files required for processing includes:

- ASN (*_asn.fits) files: These small FITS tables provide the relationship between the input exposures and the output products with the output filenames defined in the table. There will NOT be an ASN table for exposures which were taken by themselves (called 'singletons').
- RAW (*_raw.fits) files: Not processed directly, but required in order to get the intended value of the DRIZCORR calibration switch. The ASN files also only give the rootname, and with the possibility of multiple suffixes (_flt, _flc,...) for calibrated products, the code starts with the _raw files to insure that what is specified in the ASN table is actually present and has been calibrated before processing.
- FLT/FLC (*_flt.fits or *_flc.fits) files: These are the non-CTE-corrected (_flt) and CTE-corrected (_flc) calibrated exposures to be processed.

The FLT/FLC files will be the ones that actually get processed and updated with the new distortion models and WCSs, while the others allow the code to know what FLT/FLC files should be included in the processing. This allows for multiple associations of data to live in the same directory and not interfere with each other as they are re-processed. That can be useful when interested in combining data from multiple visits, for example.

Warning: Should any of these files not be available (found in the local directory), the code will raise an Exception when trying to run 'drizzlepac.astrodrizzle.AstroDrizzle' on the data. The message will indicate what file was missing with something like: **"Exception: File ie6d07ujq_flt not found."**

Calibration Switches

This processing serves as an official calibration step defined for HST data through the use of the **DRIZCORR** header keyword. This keyword can be found along with all the other calibration switches in the PRIMARY header (extension 0) of the exposures FITS file. A quick way to view this (or any keyword) value would be with:

```
from astropy.io import fits
val = fits.getval('ie6d07ujq_flt.fits', 'drizcorr')
```

This switch must be set to 'PERFORM' in order to allow the processing to be done. Processing will be completely skipped should the value of this switch in the '_raw.fits' file be set to 'OMIT'.

Log Files

A number of log files, or 'trailer' files, get generated during processing, and their filenames get defined as early as possible in the processing. The primary file will be a file with a '.tra' extension and should have the same '<rootname>' as the input file used to start the code. For example, if you were to reprocess 'ie6d07030_asn.fits', you would end up with a trailer file with the name 'ie6d07030.tra'.

This log file contains the messages generated from performing all the updates to the distortion model, updates from the astrometry database (if any), and all the image combinations performed by 'AstroDrizzle()' to create the final set of calibrated, drizzled exposures. Should any problems arise when during the processing, the log can provide the error messages and tracebacks to determine what went wrong.

Data to be Processed

Once the code has performed all the initialization, it prepares the processing by defining what files need to be combined together from the input files it can find. This includes looking for CTE-corrected versions of the calibrated exposures (FLC files) as well as all the non-CTE-corrected files (FLT files) and creating a separate list of each type. Many types of data do not get CTE-corrected by the instruments calibration software, such as calacs.e or calwf3.e, and so no list of FLC files will be made. This will tell the code that it only needs to process the FLT files by themselves. If FLC files are found, all updates to the astrometry and WCS will be performed on those files and the results then get copied into the FLT file headers upon completion of the processing.

Update the WCS

The first operation on the calibrated input files focuses on applying the calibrations for the distortion model to the WCS. This operation gets performed using the `updatewcs` task using the syntax:

```
from stwcs.updatewcs import updatewcs
updatewcs(calfiles_flc, use_db=False)
```

where `calfiles_flc` is the list of CTE-corrected FLC files or in the case there are no CTE-corrected files, the list of calibrated FLT files. Crucially, the use of `use_db=False` forces `updatewcs()` from stwcs.updatewcs to only apply the distortion model to the default WCS to create what is referred to as the **pipeline-default WCS**. This WCS has a `WCSNAME` associated with it that has the format `IDC_<rootname>` where <`rootname`> is the rootname of the `IDCTAB` reference files applied to the WCS.

This default WCS serves as the basis for all subsequent processing as the code tries to determine the WCS which is aligned most closely to the GAIA astrometric coordinate system.

Generate the initial default products

The instrument teams have calibrated the distortion models extremely well for nearly all imaging modes with the latest calibration model being applied to the WCS keywords when the observations were updated in the previous step. The observations at this point represent what the best calibration of the pointings as observed by the telescope. The accuracy of the guiding allows for sub-pixel alignment of the observations for most of the data and this step applies the distortion model to generate the 'pipeline-default' drizzle products.

The default products get generated using the *astrodrizzle* task. This initial run relies on a couple of default settings to generate the default drizzle products; namely,

- reads and applies default parameter settings from MDRIZTAB specified in observation header
- uses `resetbits=4096`
- runs with `crbit=4096` to define cosmic-rays/bad-pixels with DQ flag of 4096

Identify Cosmic-Rays

Generating these drizzle products serves as the initial attempt to identify and to flag bad-pixels or cosmic-rays in each of the observations. Assuming the relative alignment of the initial pointing by the telescope is good (aligned to <0.1 pixels), most of the cosmic-rays will be successfully identified at this point by flagging those pixels with a value of 4096 in the DQ array for each chip. This will make it easier to find sources and confirm alignment without having to weed through so many false sources. However, there are times when the default alignment by the telescope was not maintained which can result in all sources (real and cosmic-rays alike) to be flagged, so subsequent steps can reset the DQ bits from 4096 to 0 while processing the data again with `astrodrizzle` using different WCS solutions.

These initial products will only be generated for the CTE-corrected versions of the observations (`*_flc.fits` or FLC files) if they are present, and the standard calibrated versions of the observations (`*flt.fits` or FLT files) otherwise.

Verifying Alignment

The relative alignment of these pipeline-default products relies entirely on the guiding accuracy of the telescope. Unfortunately, there are times when guiding problems impact the observations. These guiding errors can occur due to any of several reasons, including but not limited to:

- re-acquisition of a different guide star from one orbit to another, usually as a result of using a close binary that was not previously identified in the guide star catalog
- high slew rate due to only guiding on gyros due to problems with acquiring guide stars
- spurious guiding problems due to the aging telescope and guiding systems

Computing the Focus Index

Verifying whether or not we can identify any problems with the relative alignment for these products starts by measuring the focus index for the drizzled products. The focus index was based on using the properties of the Laplacian of Gaussian (LoG) operator as an edge detector. See http://alumni.media.mit.edu/~maov/classes/

vision09/lect/09_Image_Filtering_Edge_Detection_09.pdf for background on the Laplacian of Gaussian operator and its use in image filtering. The index that has been implemented is based on the maximum value of the LoG operation on each drizzled product.

The process for computing this index is:

- use the drizzled product, with as many cosmic-rays removed as possible, as the input
- mask out all the saturated sources as well as possible
- apply the LoG operator to the image
- pick out the pixel with the maximum value to serve as the value of the focus index

This measurement process gets applied to the total drizzle product for an association, as well as the drizzle product for each input exposure as well, known as 'single drizzled' products. The single drizzled products represent the optimal focus since there is only a single exposure with only telescope focus changes affecting the image focus value. The range of values from the single drizzled products establishes the distribution of 'good' focus values that gets used to evaluate whether the total drizzle product passes focus verification. This range of values comes as a result of the changing focus of the telescope from one exposure to another and to a lesser extent the effect of noise in low-S/N observations.

A Z-score then gets computed for the focus index value of each single drizzle product. In simplest terms, the Z-score is a measure of how many sigma above or below the population mean a measured valued is. The actual computation is:

```
from scipy.stats as st

p = st.norm.cdf(x=val, loc=mean, scale=sigma)
z_score = st.norm.ppf(p)
```

A Z-score then gets computed for the focus index value derived from the total drizzle product. If this score falls within the range of values defined by the single drizzle focus index Z-score values, this WCS solution is considered to have passed the 'focus verification' check.

Computing the Similarity Index

In addition to the focus index, a similarity index can also be computed between the single drizzle products (again treated as 'truth') and the total drizzle product. The function used to compute this is the `max_overlap_diff` function in *astrometric_utils*. The similarity index gets computed only for the region of maximum overlap of all the input exposures. This region of overlap gets determined using the `SkyFootprint` class from the *cell_utils* module. Should an input exposure not overlap the regions where most of the exposures overlap, then the region which overlaps at least 1 other exposure will be used for computing the index.

Point sources are detected in the selected region of overlap with a mask being generated for each source containing a value of 1 for the point source and 0 for the background. The sources are identified in the single drizzle image overlap region and the total drizzle product overlap region. These single drizzle mask then gets subtracted from the total drizzle mask, then scaled by the number of non-zero pixels in the single drizzle mask resulting in a Hamming distance between the two images. The Hamming distance, simply put, provides the percentage of differences pixel-by-pixel between two arrays as described in the scipy package spatial.distance. This distance then gets scaled by the relative exposure time of the single drizzle image to

account for uncertainties introduced by readout noise, low S/N detection of sources and other variances due to exposure time.

We then compute a variant of the Mean Squared Error (MSE) algorithm used in the AmphiIndex image comparison code used for comparing images taken of amphibians. One description of how the MSE measures the similarity between images can be found at https://www.pyimagesearch.com/2014/09/15/python-compare-two-images/. This similarity index is sensitive to small offsets between exposures, as well as differences in noise, overall S/N, and even presence of cosmic-rays. In contrast, the Hamming-distance is not as sensitive to noise. Therefore, we compare the MSE similarity with the Hamming distance and take the minimum of the two values as a more robust measure of the similarity of the images. Both values share one key characteristic: values > 1.0 indicate more pixels are different than similar. The code takes the maximum value of the similarity indices computed for the total drizzle product compared to all the single drizzle products as the final measure of the similarity. If this value is less than 1.0, then this WCS is considered to have passed the similarity check.

Updating the Trailer File

Associations where there are problems with the alignment will cause this verification to fail since the sources will not be 'as sharp' based on the LoG operator. As a result, it can flag situations where even sub-pixel offsets down less than 0.5 pixels are identified. For the default pipeline alignment, failure at this point is only noted in the log with the hope that later alignment efforts will resolve the problem affecting the original input data as noted in this check.

Applying A Priori WCS Solutions

A priori WCS solutions defined for use with HST data refer to improvements to the WCS solutions that were pre-computed. As of 2020, there were 2 primary sources of *a priori* WCS solutions:

- **GSC240**: correcting the previous guide star coordinates to the GAIA frame
- **HSC30**: corrections derived using the Hubble Source Catalog(HSC) coordinates cross-matched to the GAIA catalog

The updated *a priori* solutions are stored as `headerlets` in an astrometry database. The headerlet format allows them to be applied directly to the exposure using the STWCS package while requiring very little storage space (typically, < 120Kb per headerlet). More details on the `headerlet` can be found at https://stwcs.readthedocs.io/en/latest/headerlet.html.

These solutions get applied through the use of the updatewcs task in STWCS. This task not only recomputes the PRIMARY WCS (one used by DS9 for coordinates), but also queries the astrometry database to append all additional updated WCS solutions as headerlet extensions based on the IDCTAB specified in the image header. The astrometry database may also have solutions based on additional IDCTAB solutions, but those will only be applied if `updatewcs` gets run manually with a non-default value for the `all_wcs` parameter.

In the process of modifying the file, `updatewcs` also insures that there are no duplicate solutions based on the `HDRNAME` keyword unless otherwise specified by the user. Duplicate solutions can come from any source, even inadvertantly by the user when performing image alignment on their own, so removing duplicates insures that the file does not get cluttered with unnecessary extensions. This also highlights the need to insure that all new WCS solutions get provided with unique `HDRNAME`, and preferably `WCSNAME` also, keyword values. This

will insure that the headerlet module does not thrown an Exception when trying to work with these alternate WCS headerlet extensions.

Astrometry Database

A publicly accessible database has been established to serve as a repository of *a priori* WCS solutions (full descriptions of which are found in the following sections) as well as pipeline-generated *a posteriori* WCS solutions. This database can be accessed through functions provided by the STWCS updatewcs.astrometry_utils module. This can result in several WCS solutions being available for each exposure, with one set of solutions for each distortion model that has been in use for these instruments since we initialized the database in early 2019. The functions in the stwcs.updatewcs.astrometry_utils module will allow someone to determine the full list of WCSs available for a given exposure and have them applied as desired to a given exposure.

Supporting New IDCTABs

Calibrations of the distortion model for each instrument evolves over time due to changes in the telescope as well as improvements in the modeling of the distortion, including better understanding of the time-dependent aspects of the distortion model. These new models get provided as new versions of the `IDCTAB` reference file, along with the `D2IMFILE` and `NPOLFILE`. The astrometry database contains *a priori* WCSs which represent the WCS for each exposure based on the coordinates of the guide stars used for the exposure after updating their coordinates to ones determined from the GAIA catalogs. However, they were originally computed based on the `IDCTAB` reference file in use when the database was first established.

If the `IDCTAB` specified in the image is not found in any of the WCSs in the database, the *a priori* WCS based on that `IDCTAB` get determined by the `stwcs.updatewcs.updatewcs()`. It starts by querying the guide-star web interface to retrieve the corrections from the original guide star coordinates using the stwcs.updatewcs.astrometry_utils.find_gsc_offset function. These offsets can also evolve as new GAIA catalogs are released to provide more accurate coordinates for the guide stars. These offsets are then used to correct the reference point of the pipeline-default WCS based on the new `IDCTAB` using the `apply_new_apriori` method in the AstrometryDB class. This method uses the same lines of code used to populate the astrometry database with the original set of *a priori* WCS solutions. This not only insures that there is always a GAIA-based WCS available for all exposures, but it does it using the best available information. This new *a priori* WCS not only gets added to the image as an alternate (and perhaps PRIMARY) WCS, but it also gets written out as a headerlet as an archive of the new WCS.

GSC240: GAIA and the HST Guide Stars

Observations taken prior to October 2017 used guide star coordinates which were based on guide star coordinates derived primarily from ground-based observations. This resulted in an uncertainty of 1 arcsecond in the absolute pointing of the telescope for any given observation. The development and availability of the space-based GAIA astrometric catalog finally allowed for the guide star coordinates to be known to better than 10 milli-arcseconds in 2015 with proper motion uncertainties increasing by 5 milli-arcseconds per year on average. The GAIA astrometric catalog was then cross-matched to the HST guide star catalog used for pointing the telescope, and corrections were determined. These corrections were then applied to every HST

observation taken before Oct 2017 as if the telescope used the GAIA coordinates originally to generate updated WCS solutions to describe the GAIA-based pointing. These updated WCS solutions were labelled with 'GSC240' in the WCSNAME and stored in an astrometry database to be applied on-demand to all observations taken before Oct 2017.

These solutions will not result in perfect alignment to the GAIA catalog, due to temporal uncertainties in the calibration of the instrument's field of view relative to the FGS's used to point and to guide the telescope during the observations. This uncertainty can be up to 0.5 arcseconds, but it still represents a significant improvement in the absolute astrometry from the 1-sigma of 1 arcsecond for previous WCS solutions.

All observations taken after Oct 2017 already used guide-star coordinates based on GAIA, so no new WCS was needed as it would simply be the same as the pipeline default WCS. However, if `stwcs.updatewcs.updatewcs()` computes the new *a priori* WCS on-the-fly for a new IDCTAB for observations taken after Oct 2017, it will be given the 'GSC240' (or newer) label in the `WCSNAME` to indicate the type of WCS being applied to the image.

HSC30: Hubble Source Catalog WCSs

The Hubble Source Catalog(HSC) (https://archive.stsci.edu/hst/hsc/) developed a comprehensive catalog of a majority of the sources observed in Hubble data. This catalog was then cross-matched to the GAIA catalog to determine improved positions for those sources. By using the updated positions from Version 3.0 of the HSC and comparing them to the original positions based on the pipeline default WCS solutions, updates were derived for all observations with sources from the HSC. The updates were then used to recompute the WCS solutions for those observations which were labelled as 'HSC30' in the WCSNAME and stored in the astrometry database.

Separate Directories

One mechanism used to enable comparisons of various WCS solutions is to keep copies of the observations with different types of WCS solutions in separate directories. Up until this point in the processing, the data has been processed in the directory where the processing was started. In order to keep the *a priori* solutions separate, a sub-directory gets created with name based on the association table rootname or the rootname of the single exposure being processed using the convention: `<rootname>_apriori`. All the FLC (or FLT, if no FLC files are present), and ASN file (if processing an association) are copied from the main directory into the new sub-directory and the process moves to the sub-directory to continue its processing.

Applying A Priori Solutions

Application of *a priori* WCS solutions computed in previous STScI automated calibration (pipeline) processing also occurs when running the `updatewcs` task with `use_db=True` (the default setting). This queries the astrometry database and retrieves the headerlets for all the *a priori* solutions. Only those WCSs based on the currently specified IDCTAB will be retained unless the user requests that all solutions be kept.

The database reports what solution is flagged as the `best` solution, which will typically result in the closest alignment to GAIA and will be the previously computed *a posteriori* solution if available. All the retrieved headerlets get appended as new extensions to the observations FITS file, then the database WCS solution

flagged as `best` gets applied to replace the active or primary WCS in the observation after saving a copy of the original primary WCS. However, this solution only gets used to replace the current PRIMARY WCS in the SCI header if it was based on the same IDCTAB as currently specified in the image primary header. The other solutions returned by the database are retained, but not applied at this point to enable the user to switch between them later as appropriate for their work.

When performing the standard processing with `runastrodriz`, we are only interested in seeing whether there are any issues in applying the pre-defined *a priori* corrections, while also setting the standard for the relative alignment for comparison with any new *a posteriori* fit that may be determined later in the processing.

Generating A Priori Products

The FLC images updated with the *a priori* WCS solutions now get combined using `astrodrizzle`. If the pipeline default focus verification succeeded, then `resetbits` will be set to 0 so that the previous DQ flags can be used. If the verification failed, though, `resetbits` gets set to 4096 so that the cosmic-rays can be identified and flagged fresh based on the alignment provided by the *a priori* WCS solutions.

This processing will result in a total combined drizzle product based on the `a priori` solution.

Evaluating Alignment

Confirming that the relative alignment between the images in the association was maintained with the *a priori* WCS now can be done. Although the *a priori* WCS solutions are vetted for accuracy, HST has taken a few hundred thousand different exposures in dozens of configurations and not all of those exposures were taken exactly as planned. Therefore, considerable effort goes into trying to verify that the alignment between the images has been maintained.

This verification starts by computing the focus index and similarity values for the total drizzle product and the single drizzle products using the same code used to verify the pipeline default WCS drizzle product. It then extends to include computing the similarity index between the *a priori* drizzle products and the pipeline default drizzle products. This will attempt to measure whether or not the *a priori* alignment is significantly different than the presumably good pipeline default alignment. Once again, if the similarity index is less than 1, the *a priori* alignment is considered to be successful.

Keeping the A Priori Alignment

Should all the verification steps indicate a successful alignment, the *a priori* WCS solution should be retained as an improved WCS solution over the pipeline default WCS. This gets done by simply copying the calibrated images which have been updated with the WCS solution (both the FLC and FLT images) from the `<rootname>_apriori` sub-directory to the main processing directory. This will replace the FLC and FLT files with the pipeline default solutions so that should no other WCS prove to be better, the *a priori* WCS solution will end up being used to generate the final drizzle products which get archived and provided to the end-user.

Performing An A Posteriori Alignment

The ultimate goal of this processing would be to have the input observations aligned as closely to an astrometric standard coordinate system as much as possible. The highest quality, highest precision astrometric catalog available would be the GAIA astrometric catalog and this processing seeks to align HST observations as closely to that catalog's coordinate system.

The *a priori* solutions provide an update to the astrometry based on either the guide stars used (the `GSC240` and related solutions) or manually verified alignment of sources from the observations field-of-view performed using the Hubble Source Catalog (the `HSC30` solution). Unfortunately, both of these types of solutions fail to account for sources of astrometric error which can still affect the observations and result in offsets from the GAIA system due to updates in the distortion calibration for the instruments or uncertainties in the position of the detectors field-of-view relative to the Fine Guidance Sensors (FGS) and the guide stars used for taking the observaitons.

The only way to correct for those effects remains to identify sources from the observations and perform a fit to the GAIA catalog directly. This is called an `a posteriori` solution when it can be done successfully. However, this can only be performed for observations which contain enough detectable sources, specifically sources found in the GAIA catalog. Not all observations meet this criteria either due to exposure time (too long or too short), wavelength of observation, filter bandpass (narrowband vs wide-band) and even number of sources in the field. This processing code makes no assumptions about the possibility of success and tries to perform this *a posteriori* fit on all observations.

Copying the Observations

Copies of the observaions are made in a sub-directory named after the input file used to start the processing with the convention:

<rootname>_aposteriori

For example, if the association **icw402010_asn.fits** was being processed, this directory would be named **icw402010_aposteriori**.

All the calibrated FLC and/or FLT images along with the ASN file are copied into this sub-directory. These files, at this point, have the best available WCS at this time which is most likely an *a priori* solution. This improves the chance that the *a posteriori* fit will work by minimizing the offset from GAIA which needs to be searched to find a cross-match with the GAIA sources in the field-of-view.

Aligning the Observations

The alignment process gets performed using the *perform_align()* function from the *align* module. This function performs the following steps in an attempt to perform an *a posteriori* fit to GAIA:

- Evaluates all the input observations to identify any which can not be aligned, such as GRISM or SCAN mode observations. For a full description of all the type of observations that can be filtered out, see *haputils.analyze*.
- Compute a 2D background for all the observations using `photutils`
- Determine a PSF kernel from the detectable sources in the image, if possible.

- Segments the image after applying the 2D background to identify as many sources as possible above a threshold using photutils.segmentation
- Performs source centering using photutils.detection.DAOStarFinder
- Keeps the position of the single brightest source nearest the center of the segment as the catalog position for each segment's object.
- Checks whether there are enough sources to potentially get a viable linear fit.
  - If not, the attempt at an *a posteriori* fit quits without updating the WCS of the input files.
- Queries the GAIA DR2 catalog through the STScI web service to obtain a catalog of GAIA sources that overlap the field-of-view of the combined set of observations. This catalog will serve as the **reference catalog** for the fitting process.
  - If there are not enough GAIA sources overlapping these observations, then the fit attempt quits without updating the WCS of the input files.
- Provide the source catalogs for each input image, each input images's WCS, and the GAIA reference catalog to function `align_wcs()` in the `tweakwcs` package.
  - This function cross-matches the source catalog from each image with the GAIA catalog and performs an **rscale** linear fit (as defined by `runastrodriz`), then updates the input WCS with the results of the fit upon success. See the tweakwcs readthedocs pages for more details.
  - The function `align_wcs` is first called without using the GAIA reference catalog in order to perform a relative alignment between the observations.
  - The function `align_wcs` is then called with the GAIA catalog as the reference in order to finally perform a single fit to the GAIA catalog for all the observations at the same time.
- Evaluate the success/failure state of the fit and the quality of any successful fit.
- Repeat the fit with `align_wcs` with other GAIA catalogs; including GAIA DR1 or any others specified for use in `runastrodriz` itself.
- Select the fit to the GAIA catalog which results in the lowest RMS.
  - Some fields are dominated by external galaxies with no proper motion for which GAIA DR1 without proper motions provides the best fit (lowest RMS).
  - Other fields are dominated by local galactic stars with appreciable proper motions best accounted for (still with some error) by the GAIA DR2 catalog with its proper motions.
- Keep the WCS's updated with the **best** solution and update the **WCSNAME** keyword for those WCSs to reflect the type of fit that was successful and the catalog that was used.
  - The naming convention is more fully described on the Drizzlepac Astrometry description.

The result of this lengthy process is a set of WCS objects which have been updated with a fit to a GAIA catalog representing an *a posteriori* solution.

Generate the Aligned Drizzle Products

Successful alignment of the WCSs to a GAIA catalog means that these *a posteriori* updated exposures can be combined to create a drizzled product using `AstroDrizzle`.

Verify the A Posteriori Alignment

These newly updated drizzle products still need to be evaluated to insure that the fit performed to GAIA maintained relative alignment between the images as well. Mis-alignment of the images to each other can result from too few sources being used for the fit imprinting the errors in those source positions on the relative alignment. The verification used is the same focus and similarity checks that were performed on the *a priori* updated drizzle products and even the pipeline default drizzle products.

A Posteriori Failure

At any number of points throughout this computation and verification, it could end up quitting and flagging this attempt as a failure. If this happens, no updated WCS solutions get created or saved and processing returns to the parent directory while deleting the entire `<dataset>_aposteriori` directory along with all the mis-aligned or un-alignable files. This allows the processing to revert to using the previously verified WCS solutions as the `best` WCS solution available for these observations.

A Posteriori Success

Successfully fitting to GAIA can only be declared after the verification process returned values indicating good alignment in the drizzle product. The processing would then copy these *a posteriori*-updated input exposures from the sub-directory these computations were being performed in based up to the parent directory to replace the previously updated versions of the input files. This entire sub-directory then gets deleted, unless the processing was being run in debug mode.

Creation of Final Aligned Products

The starting directory now contains updated input FLC/FLT files based on WCSs which have been verified to have maintained relative alignment and with alignment as close to the GAIA astrometric coordinate system as possible. These exposures get processed by `AstroDrizzle` to create the final, combined drizzle products for the user and for archiving at STScI in the Mikulski Archive for Space Telescopes (MAST). These products include the calibrated drizzle(DRZ) products as well as any CTE-corrected drizzle(DRC) products depending on what input exposures are available.

3.1.3 Astrometry and Headerlets

The astrometry for any given observation relies upon accurate pointing information from the telescope. However, HST has evolved over time since it was put into orbit, with the focus changing over time as the telescope desorbs. The instruments have also changed positions over time relative to the FGS guides, and the coordinates for the guide stars were orginally determined using ground-based information. All this has limited the calculation of the pointing of any given observation on the sky (absolute astrometry) to no better than 1-2 arc-seconds.

Calibration of the distortion model for each instrument has been done well enough to allow observations to be combined (relative astrometry) with an accuracy of better than 5 milli-arcseconds. This has made the absolute astrometry inaccuracy stand out even more, making it more difficult to compare observations taken at different times or to compare HST data with observations from other telescopes (like Chandra).

Therefore, multiple efforts have been undertaken to improve the absolute astrometry to match the accuracy of the relative astrometry. These efforts have resulted in multiple new world coordinate systems (WCS) solutions to be developed for HST observations, starting with ACS, WFC3, and WFPC2. Complete details of the results of calibrating the distortion models and astrometry for each instrument can be found in each of the instruments web pages; namely,

- **ACS**: ACS Distortion
- **WFC3**: WFC3 Data Handbook Chapter 4: WFC3 Images: Distortion Correction and AstroDrizzle

Each solution has its own advantages and errors, making some good for one use but inadequate for others. As a result, **all new WCS solutions which are approved by STScI** are being offered with HST data provided by MAST with headerlets serving as the mechanism for providing and applying all WCS solutions.

Pipeline Astrometric Calibration

Pipeline Astrometric Calibraion

WCS Solutions

All calibrated HST products delivered by the Barbara A. Mikulski Archive for Space Telescopes (MAST) contain the best available WCS solutions available at the time the data was last processed. Unfortunately, only 1 WCS can be used at a time to transform the position in the image to an undistorted position on the sky. The FITS Standard, however, describes how multiple WCS solutions can be defined for an image in FITS Paper I (Greisen, E. W., and Calabretta, M. R., Astronomy & Astrophysics, 395, 1061-1075, 2002).

This standard defines a keyword, WCSNAME, which serves as the label for the WCS specified in the header of the observation. It also defines additional keywords, WCSNAME[A-Z], that represent additional WCS solutions which are specified as alternate WCS solutions in the same header as the active WCS associated with WCSNAME. Headerlets rely on the use of the WCSNAME keyword to manage the multiple solutions that may be defined for a given observation, with each WCS having it's own unique WCSNAME value. Headerlets also build the HDRNAME keyword which specifies the exposure name as well WCS to insure that this WCS will only ever be applied to this exposure and no other.

The use of headerlets extends this standard by allowing each new WCS, specified using the FITS Paper I standards along with the SIP convention, to be stored separately as a new extension at the end of the image's

FITS file. This can be seen in the extensions listed by astropy.fits for a sample ACS/WFC image which includes 6 additional WCS solutions from the astrometry database.:

```
No.    Name      Ver    Type      Cards   Dimensions   Format
  0  PRIMARY       1 PrimaryHDU     279   ()
  1  SCI           1 ImageHDU       201   (4096, 2048)   float32
  2  ERR           1 ImageHDU        57   (4096, 2048)   float32
  3  DQ            1 ImageHDU        49   (4096, 2048)   int16
  4  SCI           2 ImageHDU       199   (4096, 2048)   float32
  5  ERR           2 ImageHDU        57   (4096, 2048)   float32
  6  DQ            2 ImageHDU        49   (4096, 2048)   int16
  7  D2IMARR       1 ImageHDU        15   (64, 32)   float32
  8  D2IMARR       2 ImageHDU        15   (64, 32)   float32
  9  D2IMARR       3 ImageHDU        15   (64, 32)   float32
 10  D2IMARR       4 ImageHDU        15   (64, 32)   float32
 11  WCSDVARR      1 ImageHDU        15   (64, 32)   float32
 12  WCSDVARR      2 ImageHDU        15   (64, 32)   float32
 13  WCSDVARR      3 ImageHDU        15   (64, 32)   float32
 14  WCSDVARR      4 ImageHDU        15   (64, 32)   float32
 15  WCSCORR       1 BinTableHDU     59   14R x 24C   [40A, I, A, 24A, 24A, 24A,␣
↪24A, D, D, D, D, D, D, D, D, 24A, 24A, D, D, D, D, J, 40A, 128A]
 16  HDRLET        1 HeaderletHDU    22   ()
 17  HDRLET        2 HeaderletHDU    22   ()
 18  HDRLET        3 HeaderletHDU    26   ()
 19  HDRLET        4 HeaderletHDU    26   ()
 20  HDRLET        5 HeaderletHDU    26   ()
 21  HDRLET        6 HeaderletHDU    26   ()
```

In this example, extensions 16 through 21 contain headerlets each of which specifies different astrometric alignment (each may potentially be based on a different distortion model) and each has been assigned a unique name given by the **WCSNAME** keyword. A quick listing of the names of all these WCSs can be obtained using the Python library STWCS:

```
from stwcs.wcsutil import headerlet
headerlet.get_headerlet_kw_names("j95y04hq_flc.fits", kw='WCSNAME')
['OPUS',
'IDC_0461802ej',
'OPUS-GSC240',
'IDC_0461802ej-GSC240',
'OPUS-HSC30',
'IDC_0461802ej-HSC30']
```

Interpreting WCS names

The first WCS solution listed, OPUS, corresponds to the default WCS defined by the HST calibration pipeline based on original telemetry and only a first-order distortion model. Solutions which contain **IDC_** are based on the distortion model based on the reference data associated with the **IDCTAB** reference file whose root-name is specified in the WCSNAME. For example, **IDC_0461802ej** refers to a WCS solution based on the **IDCTAB** reference file 0461802ej_idc.fits along with all the associated secondary distortion reference files given by **DGEOFILE**, **NPOLFILE** and, for WFPC2, **OFFTAB** and **DXYFILE**.

Additional WCS solutions will then be based on either the original **OPUS** WCS or the distortion-corrected **IDC_** WCS solutions. Two types of solutions can be defined for images; namely, *a priori* and *a posteriori* solutions.

a priori solutions

The *a priori* solutions have been determined for **ALL HST data** by correcting the coordinates of the guide stars that were used from the originally specified coordinates to the coordinates of those guide stars as determined by GAIA. The naming convention for these *a priori* solutions are:

```
<Starting WCS>-<Astrometric Catalog>

For example,
'IDC_0461802ej-GSC240'
```

where the **Astrometric Catalog** refers the exact astrometric catalog used to correct the guide star positions. A number of **Astrometric Catalogs** are available through MAST for aligning images. Solutions generated for the database were initially based on catalogs which were based on the GAIA catalog. These GAIA-based catalogs include:

- **GSC240**
  - This catalog contains version 2.4.0 of the *Guide Star Coordinates* (GSC) catalog,
  - All guide stars in the catalog were cross-matched with the GAIA DR1 catalog and corrected to the coordinates reported in GAIA DR1.
  - **APPLIES TO**: All HST datasets which had a successful guide star acquisition, which is nearly all data in the archive.
  - **NOTE**: HST Observations taken after September 2017.
- **HSC30**
  - This catalog contains version 3.0 of the *Hubble Source Catalog* (HSC)
  - Technically, this is an *a posteriori* solution, but it is applied blindly without further verification that the correction fully aligns the image to GAIA;hence, it is included as an *a priori* solution.
  - Sources in the HSC were cross-matched with the GAIA DR1 catalog.
  - Those cross-matched sources were then used to determine a fit to the GAIA catalog.
  - The fit to GAIA was then applied to all remaining sources in the catalog.

  - **APPLIES TO**: Only datasets which had a sufficient number of sources in the exposure to be aligned to GAIA by the *Hubble Legacy Archive(HLA)* project.

- **GAIADR1**
  - A MAST-provided version of the first data release (DR1) version of the official GAIA astrometric catalog.
  - This version does not have proper motions for a majority of the sources in the catalog.
- **GAIADR2**
  - A MAST-provided version of the second data release version of the official GAIA astrometric catalog.
  - This catalog contains initial proper motion measurements (and errors) for most sources in the catalog.
- **GAIAeDR3**
  - A MAST-provided version of the early release of the third data release version of the official GAIA astrometric catalog.
  - This catalog contains improved proper motion measurements (and errors) for most sources in the catalog compared to GAIADR2.
  - This catalog also contains a larger number of sources compared to previous releases of the GAIA catalog.

Although all solutions are appended to each FITS file, only 1 WCS (referred to as the **'active' WCS**) can be used at a time to represent the transformation from pixel coordinates to world coordinates. The active WCS is defined by the standard WCS keywords found in the header of the science extension for each chip in the exposure; *e.g.*, CRVAL1, CRVAL2, CRPIX1, CRPIX2, and so on.

The *a priori* solution which gets selected to replace the active WCS solution represents the most accurate solution available in the astrometry database at the time, and will be chosen based on the following hierarchy (as of Summer 2019):

1. HSC30
2. GSC240
3. IDC_<rootname> (distortion-corrected pipeline default WCS)

If the first type of solution is not available, the next solution in the list is selected. As new solutions are added to the astrometry database, these rules will be modified to always try to return the WCS correction which best aligns the data to the GAIA catalog.

a posteriori solutions

The *a posteriori* solutions, on the other hand, get determined from measuring sources in each image, finding overlapping sources from an astrometric catalog, identifying and cross-matching image sources with sources from the astrometric catalog and performing a fit to correct the WCS. These type of solutions can not be determined for all datasets due to a number of reasons, such as lack of sources in the image and/or lack of overlapping sources from an astrometric catalog. When these solutions can be determined for an observation, they are given a value for the `WCSNAME` keyword which follows the convention:

<Starting WCS>-FIT_<REL|IMG|EVM|SVM>_<Astrometric Catalog>

For example,

'IDC_0461802ej-FIT_REL_GAIADR2'

The terms are defined as:

- **<Starting WCS>**
  - Value of WCSNAME for the exposure prior to applying any astrometric Solutions
  - IDC_<rootname> (like `IDC_041802ej`) refers to a distortion-corrected model based on the IDCTAB reference file `0461802ej_idc.fits`.
- **`FIT`**
  - This term refers to the fact that sources from the image were identified, cross-matched and fit to sources from an astrometric catalog to create an *a posteriori* WCS solution.
- **`<REL|IMG|EVM|SVM>`**
  - `REL` : This term denotes the fact that all images were aligned relative (REL) to each other and then aligned to an astrometric catalog. This attempts to maintain the original relative alignment between the images in a given visit.
  - `IMG` : This term denotes the fact the the images were fit individually to the astrometric catalog. These solutions are applied only when relative alignment does not yield a viable fit to the astrometric catalog.
  - `EVM` : The cross-match and fit to an astrometric catalog was performed on a single exposure by itself as part of processing the exposures of an entire visit. This will typically only apply to those rare visits which do not have enough valid exposures in the visit for alignment.
  - `SVM` : This term refers to alignment of all the exposures in a single-visit to an astrometric catalog. The exposures of a visit are aligned to each other (relative alignment), then, as a group, all the exposures are cross-matched and fit to the astrometric catalog specified in the next term in the WCSNAME.
- **<Astrometric Catalog>**
  - This term describes the astrometric catalog, as listed for use with the *a priori* solutions, which was used for the cross-matching and fitting sources identified in the image(s). If a value of **NONE** is specified here, it indicates that although the image

appears (according to the code) to have been successfully relatively aligned one exposure to another, there were indications that the alignment to an astrometric catalog like **GAIADR2** failed. The user will need to carefully review the state of alignment of this data when **NONE** is listed in the output WCS.

Note: As of v3.3.0, the **GAIAeDR3** catalog serves as the default catalog to use for aligning HST data.

These separate terms provide as succinct a description of the solution determined for and applied to the exposure as possible. Additional keywords have been written out to the headerlet extension for the *a posteriori* fit which further describe the solution, including:

- number of sources used in the fit
- RMS in RA and Dec of the fit
- parameters determined for the fit
- and more…

Note: A successfully determined *a posteriori* solution will **always** be used to replace the active WCS (after insuring the previous WCS has been saved as a headerlet extension already) regardless of the original solution.

Pipeline Processing

All HST observations get processed in an automated environment using standard parameters for the calibration code, including the alignment and combination of individual exposures into undistorted products. The standard pipeline processing to create the undistorted drizzled images (drc.fits or drz.fits) gets performed using the 'runastrodriz' task in this package. This same processing can be run at any time using:

```
runastrodriz j8cw03010_asn.fits

runastrodriz j8cw03f6q_raw.fits
```

The files which need to be present are:

- RAW files (*raw.fits)
- FLT files (*flt.fits)
- FLC files (*flc.fits, if any were created by the pipeline)
- ASN file (*asn.fits, if applicable)

This processing includes a lot of logic intended to not only apply pre-defined (apriori) WCS solutions, but also to try and determine a new aposteriori solution then verify which solution (default pipeline, apriori or aposteriori) actually provides the WCS which comes closest to the GAIA astrometric frame. The runastrodriz task provides the full discussion of the logic used to define the defined 'active' WCS that gets used to create the products which get archived.

Choosing a WCS

The **only** WCS solution that gets used to perform coordinate transformations on the pixel values will be the 'active' or 'primary' WCS associated with the WCSNAME keyword. The pipeline generated products will include an active WCS which the pipeline specifies as the *best* available WCS given the information used at the time of processing. However, this default 'active' WCS may not be appropriate for all science, so this WCS may need to be replaced by one of the other WCSs instead to best support the analysis necessary for the research.

Dependent Packages

Working with the WCS solutions and headerlets gets performed using STWCS package. Examples of how to work with this package will assume that the user has already installed this package into their working Python environment and has started a Python shell. In addition, the following example relies on the Astropy IO package to work with the FITS headers and extensions.

Finally, the example described here will rely on additional functionality included in the V3.2.0 or later of the Drizzlepac package. These new functions support the generation of drizzle combined products which have been aligned to an astrometric standard catalog such as GAIA DR2.

Sample Session

This example will work with 4 exposures taken using ACS/WFC as the association *j95y04010*, with most of the example being performed on the single exposure *j95y04hpq_flc.fits*.

All the necessary libraries for working on this example can be imported using:

```
from drizzlepac.haputils import astroquery_utils as aqutils
from drizzlepac.haputils import astrometric_utils as amutils
from astropy.io import fits
from stwcs.wcsutil import headerlet
```

The data can be obtained from MAST with astroquery using a simplified interface developed in **drizzlepac** using the commands:

```
filenames = aqutils.retrieve_observation('j95y04010')
# filenames = ['j95y04hpq_flc.fits', 'j95y04hqq_flc.fits',
#              'j95y04hsq_flc.fits', 'j95y04huq_flc.fits']
```

The default 'active' WCS can be determined using:

```
default_wcsname = fits.getval(filenames[0], 'wcsname', ext=1)
```

For this example, we find that **default_wcsname='IDC_0461802ej'**, or as noted earlier, the default distortion correction based WCS provided by the pipeline with no correction for any astrometric catalogs, has been designated by the pipeline as the 'active' WCS.

All available WCSs provided by the astrometry database and attached as headerlet extensions can be queried to find the WCSNAMEs for all the new WCSs using:

```
new_wcsnames = headerlet.get_headerlet_kw_names(filenames[0], kw='WCSNAME')
```

These will be the same ones listed earlier. For this example, we decide we would like to have this observation aligned using the guide stars corrected to GAIA DR1 through the use of the GSC240-based WCS; specifically, **IDC_0461802ej-GSC240**. We can replace the 'active' WCS with this new one using:

```
new_hdrnames = headerlet.get_headerlet_kw_names(filenames[0])
# identify hdrname that corresponds with desired WCS with name of IDC_041802ej-
↪GSC240
new_wcs = new_hdrnames[new_wcsnames.index('IDC_0416802ej-GSC240')]
headerlet.restore_from_headerlet(filenames[0], hdrname=new_wcs, force=True)
# confirm new WCS is now 'active'
fits.getval(filenames[0], 'wcsname', ext=1)
```

At this point, the exposure has been updated to perform all coordinate transformations with the new GAIA DR1-based WCS as if the guide stars used for taking the observation has GAIA DR1 coordinates in the first place. This will not mean it will align perfectly with GAIA, but should be within 0.5 arcseconds due to drift in the telescope field-of-view for each of the instruments relative to the FGSs.

Headerlet Primer

The headerlet file itself conforms to FITS standards with the PRIMARY header containing global information about the WCS solution and how it was determined. Separate extensions in the headerlet then contain the header keywords for specifying the WCS for each chip in the exposure or for the distortion information necessary to correct the pixel positions from the image to the un-distorted position on the sky. These solutions rely on calibration reference data that describe the distortion observed in each instrument to better than 0.1 pixels in each detector. Instead of having to retrieve separate files with this distortion information, that distortion information has been folded into the header of each WFC3, ACS and WFPC2 dataset.

Headerlet File Structure

This new object complete with the NPOLFILE and the D2IMFILE extensions derived from the full FITS file fully describes the WCS of each chip and serves without further modification as the definition of the `headerlet`. The listing of the FITS extensions for a `headerlet` for the sample ACS/WFC exposure after writing it out to a file would then be:

```
EXT#  FITSNAME      FILENAME                EXTVE DIMENS       BITPI OBJECT

0     j8hw27c4q     j8hw27c4q_hdr.fits                         16
1       IMAGE       D2IMARR                 1     4096         -32
2       IMAGE       WCSDVARR                1     64x32        -32
3       IMAGE       WCSDVARR                2     64x32        -32
4       IMAGE       WCSDVARR                3     64x32        -32
```

(continues on next page)

(continued from previous page)

```
5       IMAGE       WCSDVARR              4     64x32          -32
6       IMAGE       SIPWCS                1                    8
7       IMAGE       SIPWCS                2                    8
```

Detailed Description of Headerlets

The full details on the headerlet, it's required set of keywords, and how the distortion models get described in the headerlet can be found in the Technical Report on Headerlets.

Code Interface to Headerlets

The STWCS package provides the code used to work with headerlets and WCS solutions.

Astrometry Database (STWCS)

Astrometry Database <https://stwcs.readthedocs.io/en/latest/astrometry_utils.html>

3.1.4 Pipeline Drizzle Parameters

This module supports the interpretation of the `MDRIZTAB` for processing as used in the pipeline. The MDRIZTAB reference table contains AstroDrizzle task parameters optimized for a wide range of observations. AstroDrizzle uses this table to match the best parameter values with the type of observations being processed. Each instrument detector has its own MDRIZTAB reference table.

Authors
Warren Hack, Ivo Busko, Christopher Hanley

License
License

drizzlepac.mdzhandler.**cleanBlank**(*value*)

drizzlepac.mdzhandler.**cleanInt**(*value*)

drizzlepac.mdzhandler.**cleanNaN**(*value*)

drizzlepac.mdzhandler.**findFormat**(*format*)

drizzlepac.mdzhandler.**getMdriztabParameters**(*files*)

Gets entry in MDRIZTAB where task parameters live. This method returns a record array mapping the selected row.

drizzlepac.mdzhandler.**toBoolean**(*flag*)

3.2 Hubble Advanced Products

3.2.1 Introduction

The `drizzlepac` package can be used for many purposes, all related to aligning and combining images to create products which can provide the deepest available views of the data. Combining the data with `drizzlepac` relies on the WCS solution specified in the input image headers. These WCS solutions are expected to align the images to each other (relative astrometry) as well as align the image to the correct position on the sky (absolute astrometry). The telemetry from HST allows the relative astrometry to be known extremely accurately (sub-milli-arcsecond level) when all images use the same guide stars and when the images were taken in the same visit. However, data taken at different times using different guide stars have historically had errors in the alignment with a sigma of 1 arc-second (or more). As a result, corrections to the alignment need to be made in order to successfully combine the images.

The code being used in the automated HST calibration pipeline to generate the products served by the HST Archive through the MAST Portal relies on multiple methods to do the best job possible in aligning and combining data regardless of whether they came from the same visit (or instrument) or not.

On December 17, 2020, MAST began production of new ACS and WFC3 products in the HST data calibration pipeline: Hubble Legacy Archive (HLA)-style mosaics comprising the data from a single HST visit which are aligned to a common astrometric reference frame. These are the first of two types of **Hubble Advanced Products (HAP)** that will be produced in the HST data pipeline and made available through the MAST Discovery Portal.

Three levels of products are available as part of this release:

- Exposure level products contain data from a single HST exposure.
- Filter level products are produced from all exposures in a visit with a common filter.
- Total level products combine all exposures from a visit for a specific detector and are intended as a detection image for producing catalogs.

The **HAP Single-Visit Mosaics (SVMs)** differ from the standard HST drizzled data products, which are aligned filter-by-filter to Gaia. SVM data products, on the other hand, are all drizzled onto the same north-up pixel grid and may have improved relative alignment across filters within a given visit, enabling easy comparison of the images through multiple filters or for images to be combined to create color mosaics. When possible, sources in the images have been aligned directly to the Gaia source catalog to improve the WCS of the images. SVM data products with both relative alignment (by filter) and absolute alignment to Gaia will contain the string 'FIT_SVM_GAIA' in the 'WCSNAME' keyword in the science extension of the image header. More discussion on HAP alignment, may be found on the webpage Improvements in HST Astrometry.

Combining data across visits to create mosaics for every observed location on the sky gets performed using **HAP Multi-Visit Mosaic (MVM)** processing.

3.2.2 Single Visit Mosaics

Single-visit Mosaic Processing

Standard calibration pipeline processing focuses on aligning data taken as associations or as single exposures as closely to a standard astrometric coordinate frame as possible (see the DrizzlePac Handbook for full details). This involves alignment of very few images all taken under as identical conditions as possible; namely, detector, filter, guide stars, guiding mode, and so on. These observations were intended by the user to represent a single view of the object of interest in a way that allows for removal of as many calibration effects as possible.

Observations taken within a single visit, on the other hand, represent data intended to produce a multi-wavelength view of the objects in the desired field-of-view. Most of the time, those observations are taken using pairs of guide stars to provide a very stable field-of-view throughout the entire visit resulting in images which overlap almost perfectly. Unfortunately, this is not always possible due to increasing limitations of the aging telescope systems. The result is that an increasing number of visits are taken where observations drift and/or roll during the course of the visit or re-acquire at slightly different pointings from one orbit to the next. Whatever the reasons, data across each visit cannot automatically be assumed to align. Single-visit mosaic (SVM) processing attempts to correct these relative alignment errors and to align the data to an absolute astrometric frame so that all the data across all the filters used can be drizzled onto the same pixel grid.

Understanding the quality of the SVM products requires knowing what processing took place to generate that product, and more importantly, what limitations that processing may have had. The following sections provide the description of the single-visit processing. Definitions of a number of processing-specific terms used in this description can be found in the *HAP Glossary*.

Note: The products generated by SVM processing will be available through the archive as `HAP Products` and can be found through the MAST Portal using the search filter `Project=HAP`.

Single-Visit Products

All files processed as part of a single visit get renamed from the standard pipeline filenames into something which describes the data more clearly. The convention used for the names of these input and output files uses these components:

- **<propid>** : the proposal ID for this visit
- **<obsetid>** : the 2-digit visit ID from this proposal
- **<instr>** : 3 or 4 letter designation of the instrument used for the observations
- **<detector>** : name of the detector used for the observations
- **<filter>** : hyphen-separated list of filter names used for the observations
- **<ipppssoo>** : standard 8 character rootname for a single exposure defined by the pipeline
- **<ipppss>** : standard 6 character designation of the **<instr>/<propid>/<obsetid>** for this visit
- **dr[cz].fits** : suffix for drizzled products

- **fl[ct].fits** : suffix for pipeline-calibrated files
- **hlet.fits** : suffix for headerlet files containing the WCS solution used to create the final drizzle products/mosaics
- **<ipppss>_trl.txt** : single-visit processing log files
- **point-cat.ecsv** : suffix for point (aperture) photometric catalog products
- **segment-cat.ecsv** : suffix for segment photometric catalog products

These components get combined to create filenames specific to each type of file being processed. The following table provides a complete list of all the products created as a result of single-visit processing.

Table 1: Single-visit product filenames

Product	File Type	Filename for Files Produced
Exposure	drizzle product	hst_<propid>_<obsetid>_<instr>_<detector>_<filter>_<ipppssoo>_dr[cz].fits
	flat-field product	hst_<propid>_<obsetid>_<instr>_<detector>_<filter>_<ipppssoo>_fl[ct].fits
	headerlet file	hst_<propid>_<obsetid>_<instr>_<detector>_<filter>_<ipppssoo>_hlet.fits
	trailer file	hst_<propid>_<obsetid>_<instr>_<detector>_<filter>_<ipppssoo>_trl.txt
	preview (full size)	hst_<propid>_<obsetid>_<instr>_<detector>_<filter>_<ipppssoo>_dr[cz].jpg
	preview (thumbnail)	hst_<propid>_<obsetid>_<instr>_<detector>_<filter>_<ipppssoo>_dr[cz]_thumb.jpg
Filter	drizzle product	hst_<propid>_<obsetid>_<instr>_<detector>_<filter>_<ipppss>_dr[cz].fits
	point-source catalog	hst_<propid>_<obsetid>_<instr>_<detector>_<filter>_<ipppss>_point-cat.ecsv
	segment-source catalog	hst_<propid>_<obsetid>_<instr>_<detector>_<filter>_<ipppss>_segment-cat.ecsv
	trailer file	hst_<propid>_<obsetid>_<instr>_<detector>_<filter>_<ipppss>_trl.txt
	preview (full size)	hst_<propid>_<obsetid>_<instr>_<detector>_<filter>_<ipppss>_dr[cz].jpg
	preview (thumbnail)	hst_<propid>_<obsetid>_<instr>_<detector>_<filter>_<ipppss>_dr[cz]_thumb.jpg
Total	drizzle product	hst_<propid>_<obsetid>_<instr>_<detector>_total_<ipppss>_dr[cz].fits
	point-source catalog	hst_<propid>_<obsetid>_<instr>_<detector>_total_<ipppss>_point-cat.ecsv
	segment-source catalog	hst_<propid>_<obsetid>_<instr>_<detector>_total_<ipppss>_segment-cat.ecsv
	trailer file	hst_<propid>_<obsetid>_<instr>_<detector>_total_<ipppss>_trl.txt
	preview (full size)	hst_<propid>_<obsetid>_<instr>_<detector>_total_<ipppss>_dr[cz].jpg
	preview (thumbnail)	hst_<propid>_<obsetid>_<instr>_<detector>_total_<ipppss>_dr[cz]_thumb.jpg
	color preview (full size)	hst_<propid>_<obsetid>_<instr>_<detector>_total_<ipppss>_<filters>_dr[cz].jpg
	color preview (thumbnail)	hst_<propid>_<obsetid>_<instr>_<detector>_total_<ipppss>_<filters>_dr[cz]_thumb.jpg

Primary User-Interface

One task has been written to perform the single-visit processing: `runsinglehap`. It gets used by STScI to generate the single-visit products which can be found in the Mikulski Archive for Space Telescopes (MAST) archive. This task can also be run from the operating system command-line or from within a Python session to reproduce those results, or with modification of the input parameters, perhaps improve on the standard archived results. Full details on how to run this task can be found in the description of the task at *API for runsinglehap*.

Processing Steps

Single-visit processing performed by `runsinglehap` relies on the results of the standard astrometric processing of the individual exposures and associations as the starting point for alignment. This processing then follows these steps to create the final products:

1. Interpret the list of filenames for all exposures taken as part of a single visit and filter out images that cannot and/or should not be processed (e.g., exposure time of zero) from further processing.
2. Copy the pipeline-calibrated (FLT/FLC) files to the current directory for processing.
3. Rename the input files to conform to the single-visit naming conventions. (This step insures that the original pipeline results remain available in the archive unchanged)
4. Define what output products can be generated.
5. Align all exposures in a relative sense (all to each other).
6. Create a composite source catalog from all aligned input exposures.
7. Cross-match and fit this composite catalog to GAIA to determine new WCS solution.
8. Update renamed input exposures with results of alignment to GAIA.
9. Create each of the output products using the updated WCS solutions.

Note: It should be noted that processing is performed on a detector-by-detector basis; if a visit contains input data from *n* detectors, steps 5-9 will be executed serially *n* times to process the input images from each detector separately.

Processing the Input Data

SVM processing starts with a list of all the single exposures which were taken as part of a visit. Any associations which were defined by the proposal are ignored, since the visit itself gets treated, in essence, as a new association. The input files can be specified either using the **poller** file format used by the STScI automated processing or a file with a simple list of filenames (one filename per line).

Automated poller input file format

The automated processing performed to populate the MAST archive at STScI provides a file with the following format:

```
ic0s17h4q_flt.fits,12861,C0S,17,602.937317,F160W,IR,ic0s/ic0s17h4q/ic0s17h4q_flt.
↪fits
ic0s17h5q_flt.fits,12861,C0S,17,602.937317,F160W,IR,ic0s/ic0s17h5q/ic0s17h5q_flt.
↪fits
ic0s17h7q_flt.fits,12861,C0S,17,602.937317,F160W,IR,ic0s/ic0s17h7q/ic0s17h7q_flt.
↪fits
ic0s17hhq_flt.fits,12861,C0S,17,602.937317,F160W,IR,ic0s/ic0s17hhq/ic0s17hhq_flt.
↪fits
```

This example comes from the 'ic0s1' visit where the columns are:

1. exposure filename
2. proposal ID (numeric value)
3. program ID - ppp value from exposure filename
4. obset_id - visit number from proposal
5. exposure time of the exposure
6. filters used for the exposure, with muliple filters separated by a semicolon (e.g., F850LP;CLEAR2L)
7. detector used to take the exposure
8. location of the exposure in a local cache

Status of Input Data

The list of filenames which should be processed as a single-visit provides the raw science data for creating the new combined output products. However, these files need to be properly calibrated prior to SVM processing. Specifically, the exposures need to be:

- fully calibrated using the instruments calibration software, such as `calacs.e` for ACS and `calwf3.e` for WFC3 data. This should also include CTE-correction for the images whenever possible.
- processed using `runastrodriz` in order to apply the latest distortion model calibrations to the astrometry and to align the exposures as closely as possible to an external astrometric reference when possible.

These steps insure that the latest calibrations get applied to the data making it easier for the SVM processing to cross-match the data with minimal interference from artifacts in the data. In addition, the CTE-corrected versions of the data get used during pipeline processing in order to allow for better alignment of the exposures and to improve the photometry of the data as much as possible.

These processing steps can be verified in the input data using header keywords from the exposures

Table 2: Processing keywords

Header Keyword	Valid Values	Notes
FLATCORR	COMPLETED	Completion of basic calibration
DRIZCORR	COMPLETED	Completion of distortion calibration
WCSNAME	-FIT	Successful **a posteriori** alignment
	-HSC30	Successful **a priori** alignment
	-GSC240	Successful **a priori** alignment

The full set of possibilities for updated WCSs as reported using the **WCSNAME** keyword can be found in the description of the *Interpreting WCS names*.

As long as the input data meets these requirements, then SVM processing will have the best chance of success. Data which has not been able to be aligned successfully with an **a priori** or **a posteriori** solution can still be processed as part of a single-visit, however, the alignment may be more difficult to determine due to the larger uncertainties for HST pointing prior to October 2017.

Filtering the input data

Not all HST imaging observations can be aligned using SVM processing. Observations taken, for example, in SPATIAL SCAN mode result in sources which can not be aligned. The *haputils.analyze* module evaluates all input exposures using these header keywords for the stated rejection criteria.

Table 3: Single-visit product filenames

Header Keyword	Values Which Trigger Rejection	Explanation
OBSTYPE	(not IMAGING)	Tyically only Imaging mode data is processed with the exception of SPECTROSCOPIC Grism and Prism images
MTFLAG	T	No moving targets, WCS and background sources vary
SCAN_TYP	C or D (or not N)	Can not align streaked sources
FILTER or FILTER1, FILTER2	BLOCK	Internal calibration of SBC detector
EXPTIME	0	no exposure time, no data to align
TARGNAME	DARK, TUNGSTEN, BIAS, FLAT,	No alignable external sources in these calibration modes
	EARTH-CALIB, DEUTERIUM	No alignable external sources in these calibration modes
CHINJECT	not NONE	No alignable external sources in these calibration modes

Any observation which meets any of these criteria are flagged to be ignored (not processed). An exception has been allowed for data where the OBSTYPE keyword is equal to SPECTROSCOPIC and FILTER (or FILTER1, FILTER2) is equal to Grism or Prism. The Grism/Prism SVM FLT/FLC data are retained to

reconcile the active WCS between the Grism/Prism images and any valid direct exposures obtained with the same detector. In addition, any data taken where the FGSLOCK keyword contains 'COARSE' or 'GY' will be flagged as potentially compromised in the comments generated during processing.

All observations which are alignable based on these criteria are then passed along as a table to create the SVM products. Those inputs which can be processed are then copied and renamed using the *Single-Visit Products*. This insures that no SVM processing will affect or otherwise modify the original pipeline-processed input files. Only the SVM named input files will be updated with new SVM-aligned WCS solutions and then used to produce the drizzle products.

Defining the Output Products

The table with the set of observations which can be processed now gets interpreted. The goal is to identify what exposures can be combined to create unique products. This grouping will be used to create the **product list**. The **product list** is a Python list of `HAPProduct` objects, described in *drizzlepac.haputils.product* API docs, which represent each and every output product to be created for the visit. While the specifics of each Product class vary, representative Product instances contain:

- list of filenames for all input exposures that will contribute to the output drizzle product
- WCS for output drizzle product
- pre-defined names for all output files associated with this **Product** including:
  - drizzle-combined image
  - point-source catalog determined from the drizzle-combined image
  - segmentation-based catalog determined from the drizzle-combined image
  - astrometric catalog used to align the input exposures
  - output trailer (aka log) file recording the processing stages
- methods for:
  - determining average number of images per pixel
  - defining the final WCS
  - aligning the exposures to an astrometric reference (GAIA)
  - applying the selected parameters to `AstroDrizzle`
  - drizzling the inputs to create the output drizzle product
  - determining the source catalogs from the drizzle product

This interpretation of the list of input filenames gets performed using the code in *drizzlepac.haputils.poller_utils* by grouping similar observations. The rules used for grouping the inputs into output products result in outputs which have the same detector and filter. These output products are referred to as **filter products** defined as a `product/FilterProduct` instance.

All exposures for a single detector are also identified and grouped to define a **total product** using the `product/TotalProduct` class. This **total product** drizzle image provides the deepest available view of the field-of-view from this visit which will be used to produce the master catalog of sources for this visit.

The master catalog of source positions will be used to perform photometry on each exposure, whether the source can be identified in the exposure at that position or not. This **forced photometry** results in limits for the photometry in cases where the sources are not bright enough to be identified in a given filter.

Two separate source catalogs for each filter are also pre-defined; namely,

- a point-source catalog derived using `photutils` DAOStarFinder
- a segmentation-based catalog derived using `photutils` segmentation code

These two catalogs provide complimentary views of each field-of-view to try to highlight all types of compact sources found in the exposures.

Example Visit

For example, a relatively simple visit of a fairly bright and crowded field with 6 F555W exposures (two 15-second and four 30-second exposures) and 6 F814W exposures (two 5-second and four 15-second exposures) would result in the definition of these output products:

- a drizzled image for each separate exposure
- a WCS updated FLT/FLC image for each separate exposure
- a headerlet file for each separate exposure
- a trailer file for each separate exposure
- a single F555W product (a drizzled filter image and corresponding trailer file)
- a single F814W product (a drizzled filter image and corresponding trailer file)
- a single **total product** (a drizzled total detection image and corresponding trailer file)
- a point-source catalog for the F555W product
- a segmentation-based source catalog for the F555W product
- a point-source catalog for the F814W product
- a segmentation-based source catalog for the F814W product
- a point-source catalog for the total product
- a segmentation-based catalog for the total product

The function *drizzlepac.haputils.poller_utils.interpret_obset_input()* serves as the sole interface for interpreting either the input **poller** file which contains exposure information for a visit or a file which contains dataset names, one per line. A basic tree is created (as a dictionary of dictionaries) by this function where the output exposures are identified along with all the names of the input exposures. This tree then serves as the basis for organizing the rest of the SVM processing.

In addition to defining what output products need to be generated, all the SVM products names are defined using the *Single-Visit Products*. This insures that all the output products have filenames which are not only unique, but also understandable (if a bit long) that are easily grouped on disk.

Aligning the Input Data

All input exposures should have already been aligned either individually or by association table as close to GAIA as possible during standard pipeline calibration processing. However, each exposure or association (of exposures) can be aligned to slightly different fits or catalogs due to differences in the source objects which can be identified in each separate exposure. The primary goal of SVM processing is to refine this alignment so that all exposures in the visit for the same detector (those exposures which contribute to each **total product**) share the same WCS (pixels on the sky).

Alignment of all the exposures for a **total product** uses the same alignment code as the standard calibration pipeline. The basic steps it follows is:

- generate a source catalog for each exposure (using *haputils.astrometric_utils*)
- obtain the WCS from each exposure
- perform a relative fit between the exposures using `tweakwcs`
- obtain an astrometric reference catalog for the field-of-view
- perform a final fit of all the exposures at once to the astrometric catalog
- update each WCS with the final corrected WCS generated by `tweakwcs`

This basic process gets performed using different reference catalog and different fitting modes until it obtains a successful fit. The fitting loops over the following catalogs in order of priority:

- GAIAeDR3
- GSC242
- 2MASS

The field being fit may not have any GAIA sources, however, it may instead be dominated by extra-galactic sources measured by the PAN-STARRs project and included in the GSC242 catalog. Thus, the fit to GAIA may not result in any cross-matches due to lack of GAIA sources, so the algorithm would continue on to try to fit to the GSC242 catalog where it obtains enough cross-matches for a successful fit. This would immediately cause the fitting algorithm to end with this fit to the GSC242 catalog as the one to be used to update the WCS solutions of the input images for the visit.

While attempting to fit to each catalog, the algorithm tries fitting using the following geometries (again, in order of priority):

- rscale : full 6-parameter linear fit with skew terms
- rshift : 4 parameter linear fit with rotation and scale the same in X and Y
- shift : only fit for shift in X and Y, no rotation or scale terms

For example, should there only be 5 cross-matches between the image and the GSC242 catalog, then the 'rscale' fit would fail due to a requirement set by the code to only use 'rscale' with >=6 cross-matches. Thus, the 'rscale' fit would fail, and the algorithm would then try 'rshift' which would be successful. As a result, the GSC242 catalog fit using 'rshift' would be used to update the WCS solutions for all the input images.

The limits for performing the relative alignment and absolute fit to the astrometric catalog (defaults to **GAIAeDR3**) are lower under the expectation that large offsets (> 0.5 arcseconds) have already been removed

in the pipeline processing. This makes the SVM alignment more robust across a wider range of types of fields-of-view. The final updated WCS will be provided with a name that reflects this cross-filter alignment using **-FIT_SVM_<catalog name>** as the final half of the **WCSNAME** keyword. More details on the WCS naming conventions can be found in the *Interpreting WCS names* section.

Creating the Output Products

Successful alignment of the exposures allows them to be combined into the pre-defined output products; primarily, the **filter products** and the **total products**. These products get created using *drizzlepac.astrodrizzle.AstroDrizzle()*.

Selecting Drizzle Parameters

Optimal parameters for creating every possible type of output product or mosaic would require knowledge of not only the input exposures, but also expert knowledge of the science. Parameters optimized for one science goal may not be optimal for another science goal. Therefore, automated pipeline processing has defined a basic set of parameters which will result in a reasonably consistent set of products as opposed to trying to optimize for any specific science case.

The default parameters have been included as part of the `drizzlepac` package in the `drizzlepac/pars/hap_pars` directory. Index JSON files provide the options that have been developed for selecting the best available default parameter set for processing. The INDEX JSON files point to different parameter files (also in JSON format) that are also stored in sub-directories which are organized by instrument and detector.

Selection criteria are also listed in these Index JSON files for each step in the SVM processing pipeline; namely,

- alignment
- astrodrizzle
- catalog generation
- quality control

Initially, only the **astrodrizzle** step defines any selection criteria for use in processing. The criteria is based on the number of images being combined for the specific instrument and detector of the exposures.

The SVM processing interprets the input data and verifies what input data can be processed. At that point, the code determines what selection criteria apply to the data and uses that to obtain the appropriate parameter settings for the processing steps. Applying the selection to obtain the appropriate parameter file simply requires matching up the key in the JSON file with the selection information. Depending on the detector, selection information can take the form of the number of input observations, the date that the observations were taken, the central filter wavelength, or the dispersive element type. For example, a **filter product** would end up using the **filter_basic** criteria, while an 8 exposure ACS/WFC association would end up selecting the **acs_wfc_any_n6** entry.

User-customization of Parameters

The parameter configuration files now included in the `drizzlepac` package are designed to be easily customized for manual processing with both `runastrodriz` (pipeline astrometry processing) and `runsinglehap` (SVM processing). These ASCII JSON files can be edited prior to manual reprocessing to include whatever custom settings would best suit the science needs of the research being performed with the data. Template SVM processing pipeline parameter files populated with default values can be created using `generate_custom_svm_mvm_param_file`. For details on how these parameter files can be created, please refer to the *haputils.generate_custom_svm_mvm_param_file* documentation.

Defining the Output WCS

The SVM processing steps through the **product list** to generate each of the pre-defined products one at a time after the input exposures have all been aligned. One of the primary goals of SVM processing is to produce combined images which share the same WCS for all the data from the same detector. This simply requires defining a common WCS which can be used to define the output for all the **filter products** from the visit.

The common WCS, or **metawcs**, gets defined by reading in all the WCS definitions as stwcs.wcsutil.HSTWCS objects for all the input exposures taken with the same **instrument** in the visit. This list of **HSTWCS** objects then gets fed to `stwcs.distortion.utils.output_wcs`, the same function used by `AstroDrizzle` to define the default output WCS when the user does not specify one before-hand. This results in the definition of a WCS which spans the entire field-of-view for all the input exposures with the same plate scale and orientation as the first **HSTWCS** in the input list. This **metawcs** then gets used to define the shape, size and WCS pointing for all drizzle products taken with the same detector in the visit.

Handling Special Images

Grism and Prism images are acquired as part of a visit, in conjunction with their direct image counterparts, and classified as spectroscopic data. It is beneficial for these images to share a common WCS with the corresponding direct images from the same detector in the visit. Because the Grism/Prism data cannot be used in the alignment procedure due to the nature of the data, the best WCS solution that can be generated for these images is an **a priori** solution. An **a priori** solution has been determined for essentially all HST data by correcting the coordinates of the guide stars that were used for the observation to the coordinates of the same guide stars as determined by GAIA, in this case. The actual image pixels have not been used in the WCS determination. The WCSNAME for this **a priori** solution is of the form:

```
<Starting WCS>-<Astrometric Catalog>

For example,
'IDC_0461802ej-GSC240'
```

where the Astrometric Catalog refers to the specific astrometric catalog used to correct the guide star positions.

During SVM processing, all the WCS solutions in common to **all** of the Grism/Prism and direct images from the same detector in the visit are gathered and matched against a list of prioritized WCS solutions, where the preferred solution is of the form *IDC_?????????-GSC240* and the *IDC_?????????* represents

the particular IDCTAB reference file. Once a common WCS solution is determined, the active (aka primary) WCS solution for the Grism/Prism and direct images from the same detector is then set to this common solution. Any previously active WCS for the image that is not already stored in the image will be archived as a new WCS headerlet extension, unless the solution as identified by the HDRNAME, already exists as a headerlet.

The only SVM processing performed on or with Grism/Prism images is with respect to the potential update to a common active WCS with its corresponding direct images. These images are not used in *any* SVM processing steps. Effectively the images are only processed to an exposure level product. If the Grism/Prism images have no corresponding direct images acquired with the same detector, then the process of reconciling the WCS of the images in the visit is not done.

Ramp images are utilized only in the alignment to GAIA stage of the processing, thereby contributing to the computation of the **metawcs** of the total detection image based upon all exposures in the visit. During this process it is possible the active WCS of each Ramp exposure has been updated. The Ramp exposures, similar to the Grism/Prism images, are only processed to an exposure level product.

Drizzling

Each output product gets created using `AstroDrizzle`. This step:

- combines all the input exposures associated with the product
- uses the parameters read in from the configuration files
- defines the output image using the **metawcs** WCS definition
- writes out a multi-extension FITS (MEF) file for the drizzled image using the pre-defined name

This drizzled output image has the same structure as the standard pipeline drizzle products; namely,

- PRIMARY extension: all information common to the product such as instrument and detector.
- SCI extension: the drizzled science image along with header keywords describing the combined array such as total exposure time.
- WHT extension: an array reporting the drizzled weight for each pixel
- CON extension: an array reporting what input exposures contributed to each output pixel

The headers of each extension gets defined as using the `fitsblender` software with much the same rules used to create the standard pipeline drizzle product headers. In short, it uses simple rules files to determine what keywords should be kept in the output headers from all the input exposures, and how to select or compute the value from all the input headers for each keyword.

Unique SVM Keywords

A small set of keywords have been added to the standard drizzle headers to reflect the unique characteristics of the SVM products. These keywords are:

NPIXFRAC
Fraction of pixels with data

MEANEXPT
Mean exposure time per pixel with data

MEDEXPT
Median exposure time per pixel with data

MEANNEXP
Mean number of exposures per pixel with data

MEDNEXP
Median number of exposures per pixel with data

Defining the Footprint

The `S_REGION` keyword records the footprint of the final drizzle image as it appears on the sky as a list of RA and Dec positions ordered in a counter-clockwise manner. These positions outline only those pixels which have been observed by HST, not just the rectangular shape of the final drizzle array. This allows the archive to provide a preview of the drizzle products footprints in their all-sky map to assist users in selecting the data most suited for their search.

The computation of this keyword relies on automatically identifying all the corners of the exposed pixels from the final drizzle product, ordering them in a counter-clockwise manner relative to North up, then applying the WCS to transform those pixel positions into sky coordinates. The function *`compute_sregion()`* gets used to define the value of this keyword, and can be called directly for any FITS image.

Note: This function also gets called during standard-pipeline processing to populate the `S_REGION` keyword in all calibrated FLC/FLT files as well.

Note: The list of positions for the footprint can contain more corner positions than expected due to the sensitivity of the Harris corner detection algorithm used to identify the corners.

Skycell Information

The pipeline (ipppssoot) and SVM processing both add the SKYCELL keyword to the headers of the input FLT(C) and drizzled DRZ(C) products. This keyword includes all of the skycells that the input exposures overlap.

Note: The SKYCELL keyword values can differ between the pipeline and SVM products as the values depend on the WCSNAME of the input exposures.

Catalog Generation

SVM processing does not stop with the creation of the output drizzled images like the standard calibration pipeline. Instead, it derives 2 separate source catalogs from each drizzled **filter product** to provide a standardized measure of each visit. For more details on how the catalogs are produced, please refer to the *Catalog Generation* documentation page.

Catalog Quality Control

All detected sources are not created equal. Raw source catalogs typically contain a mix of scientifically legitimate point sources, scientifically legitimate extended sources, and scientifically dubious sources (those likely impacted by low signal-to-noise ratio, detector artifacts, saturation, cosmic rays, etc.). The last set of algorithms run by SVM processing classifies each detected source into one or more of these groups and assigns each source a classification value, known as a flag. Based on the flag value, sources that are obviously scientifically dubious are filtered out and not written to the final source catalogs. More details on this process can be found in section *2.4.2: Determination of Flag Values* of the catalog generation documentation page.

Catalog Generation

The Hubble Advanced Products (HAP) project generates two source list catalogs, colloquially referred to as the Point and Segment catalogs. Both catalogs are generated using utilities from Photutils with the Point catalog created based upon functionality similar to DAOPhot-style photometry, and the Segment catalog created with Source Extractor segmentation capabilities and output in mind.

These catalogs provide aperture photometry in the ABMAG system and are calibrated using the photometric zeropoints corresponding to an 'infinite' aperture. To convert to total magnitudes, aperture corrections must be applied to account for flux falling outside of the selected aperture. For details, see Whitmore et al., 2016 AJ, 151, 134W.

1: Support Infrastructure for Catalog Generation

1.1: Important Clarifications

As previously discussed in *Single-visit Mosaic Processing*, AstroDrizzle creates a single multi-filter, detector-level drizzle-combined image for source identification and one or more detector/filter-level drizzle-combined images (depending on which filters were used in the dataset) for photometry. The same set of sources identified in the multi-filter detection image is used to measure photometry for each filter. We use this method to maximize the signal across all available wavelengths at the source detection stage, thus providing photometry with the best quality source list across all available input filters.

It should also be stressed here that the point and segment photometry source list generation algorithms identify source catalogs independently of each other and DO NOT use a shared common source catalog for photometry.

Note: A catalog file will always be written out for each type of catalog whether or not there are any identified sources in the exposure.

1.2: Generation of Pixel Masks

Every multi-filter, detector-level drizzle-combined image is associated with a boolean footprint mask which defines the illuminated (True) and non-illuminated (False) portions of the image based upon its constituent exposures and the corresponding WCS solution. The boundary of the illuminated portion is iteratively eroded or contracted to minimize the impact of regions where signal quality is known to be degraded, and thereby, could affect source identification and subsequent photometric measurements. The erosion depth is approximately ten pixels and is done by using the ndimage.binary_erosion scipy tool. For computational convenience, an inverse footprint mask is also available for functions which utilize masks to indicate pixels which should be ignored during processing of the input data.

1.3: Detection Image Background Determination

For consistency, the same background and background RMS images are used by both the point and segment algorithms. To ensure optimal source detection, the multi-filter detection image must be background-subtracted. In order to accommodate the different types of detectors, disparate signal levels, and highly varying astronomical image content, three background computations are used as applicable. The first category of background definition is a special case situation, and if it is found to be applicable to the detection image, the background and background RMS images are defined and no further background evaluation is done.

It has been observed that some background regions of ACS SBC drizzle-combined detection images, *though the evaluation is done for all instrument detection images*, are measured to have values identically equal to zero. If the number of identically zero pixels in the footprint portion of the detection image exceeds a configurable percentage threshold value (default is 25%), then a two-dimensional background image is constructed and set to the value of zero, hence the **Zero Background Algorithm**. Its companion constructed

RMS image set to the RMS value computed for the non-zero pixels which reside within the footprint portion of the image.

If the **Zero Background Algorithm** is not applicable, then sigma-clipped statistics are computed, known as the **Constant Background Algorithm**, for the detection image using the astropy.stats.sigma_clipped_stats Astropy tool. This algorithm uses the detection image and its inverse footprint mask, as well as a specification for the number of standard deviations and the maximum number of iterations to compute the mean, median, and rms of the sigma-clipped data. The specification for the number of standard deviations and the maximum number of iterations are configurable values which are set to 3.0 and 3 by default, respectively.

At this point the Pearson's second coefficient of skewness is computed.

$$skewness = 3.0 * (mean - median)/rms$$

The skewness compares our sample distribution with a normal distribution where the larger the absolute value of the skewness, the more the sample distribution differs from a normal distribution. The skewness is computed in this context to aid in determining whether it is worth computing the background by other means (i.e., our third option of a two-dimensional background). For example, a high positive skew value can be indicative of there being a significant number of sources in the image which need to be taken into account with a more complex background.

Since the median and rms values will be used to generate the two-dimensional background and RMS images, respectively, the values need to be deemed reasonable. A negative mean or median value is reset to zero, and a minimum rms value is computed for comparison to the sigma-clipped statistic based upon FITS keyword values in the detection image header. For the CCD detectors only, a-to-d gain (ATODGN), read noise (READNSE), number of drizzled images (NDRIZIM), and total exposure time (TEXPTIME) are employed to compute a minimum rms, with the larger of the two rms values (sigma-clipped rms or minimum rms) adopted for further use. Once viable background and background RMS values are determined, two-dimensional images matching the dimensions of the detection image are constructed. Through a configuration setting, a user can specify the sigma-clipped statistics algorithm be the chosen method used to compute the background and RMS images, though the special case of identically zero background data will always be evaluated and will supersede the user request when applicable.

For the final background determination algorithm, **Conformal Background Algorithm**, the photutils.background.Background2d Astropy tool is *only* invoked if the **Zero Background Algorithm** has not been applied, the user has not requested that only the **Constant Background Algorithm** computed, and the skewness value derived using the sigma-clipped statistics is less than a pre-defined and configurable threshold (default value 0.5).

The **Conformal Background Algorithm** uses sigma-clipped statistics to determine background and RMS values across the image, but in a localized fashion in contrast to **Constant Background Algorithm**. An initial low-resolution estimate of the background is performed by computing sigma-clipped median values in 27x27 pixel boxes across the image. This low-resolution background image is then median-filtered using a 3x3 pixel sample window to correct for local small-scale overestimates and/or underestimates. Both the 27 and 3 pixel settings are configurable variables for the user.

Once a background and RMS image are determined using this final technique, a preliminary background-subtracted image is computed so it can be evaluated for the percentage of negative values in the illuminated portion of the image. If the percentage of negative values exceeds a configurable and defined threshold (default value 15%), the computation of the background and RMS image from this algorithm are discarded.

Instead the background and RMS images computed using **Constant Background Algorithm**, with the associated updates, are ultimately chosen as the images to use.

Attention: It cannot be emphasized enough that a well-determined background measurement, leading to a good threshold definition, is very crucial for proper and successful source identification.

1.3.1: Configurable Variables

Through-out this section variables have been mentioned which can be configured by the user. The values used for these variables for generating the default catalogs are deemed to be the best for the general situation, but users can tune these values to optimize for their own data.

To this end, users can adjust parameter values in the <instrument>_<detector>_catalog_generation_all.json files in the following path: /drizzlepac/pars/hap_pars/svm_parameters/<instrument>/<detector>/. Alternatively, a safer way for users to tune configuration settings is to first utilize *generate_custom_svm_mvm_param_file* to generate a custom parameter .json file. This parameter file, which is written to the user's current working directory by default, contains all default pipeline parameters and allows users to adjust any/or all of these parameters as they wish without overwriting the hard-coded default values stored in /drizzlepac/pars/hap_pars/svm_parameters/. To run the single-visit mosaic pipeline using the custom parameter file, users simply need to specify the name of the file with the '-c' optional command-line argument when using *runsinglehap* or the 'input_custom_pars_file' optional input argument when executing `run_hap_processing()` in *hapsequencer* from Python or from another Python script.

Warning: Modification of values in the parameter files stored in /drizzlepac/pars/hap_pars/svm_parameters/ is *strongly* discouraged as there is no way to revert these values back to their defaults once they have been changed.

1.4: Image Kernel

In an attempt to optimize the source detection for the specific image being processed, the software attempts to derive a custom image kernel based upon the data. The multi-filter detection image is analyzed to find an isolated, non-saturated point source away from the edge of the image to use as a template for a source detection kernel. If no suitable source is found, the algorithm falls back to the use of a two-dimensional Gaussian kernel based upon the supplied FWHM and the astropy.convolution.Gaussian2DKernel Astropy tool.

2: Point (Aperture) Photometric Catalog Generation

2.1: Source Identification Options

A number of options have been implemented within the catalog generation code in order to best match the contents of the exposure, including presence of saturated sources and cosmic-rays. The available options include:

- dao : The photutils DAOStarFinder class that provides an implementation of the DAOFind algorithm.
- iraf : The photutils IRAFStarFinder class that implements IRAF's *starfind* algorithm.
- psf [DEFAULT] : This option is a modification of DAOStarFinder which relies on a library of Tiny-Tim (model) PSFs to locate each source then uses DAOStarFinder to measure the final position and photometry of each identified source.

These options are selected through the "starfinder_algorithm" parameter in the JSON configuration files in the `pars/hap_pars` directory as used by *runsinglehap*.

2.1.1: Source Identification using DAOStarFinder

We use the photutils.detection.DAOStarFinder Astropy tool to identify sources in the background-subtracted multi-filter detection image. Here, the background computed using one of the algorithms discussed in Section 1.3 is applied to the science data to initialize point-source detection processing. This algorithm works by identifying local brightness maxima with roughly gaussian distributions whose peak values are above a predefined minimum threshold. This minimum threshold value is computed as the background noise times a detector-dependant scale factor (listed below in table 0). Full details of the process are described in Stetson 1987; PASP 99, 191. The exact set of input parameters fed into DAOStarFinder is detector-dependent. The parameters can be found in the <instrument>_<detector>_catalog_generation_all.json files mentioned in the previous section.

Table 4: Table 0: Background scale factor values used to compute minimum detection thresholds

Instrument/Detector	Scale Factor
ACS/HRC	5.0
ACS/SBC	6.0
ACS/WFC	5.0
WFC3/IR	1.0
WFC3/UVIS	5.0

2.1.2: Source Identification using PSFs

This option, introduced in Drizzlepac v3.3.0, drizzles model PSFs created using TinyTim to match the orientation and plate scale of the observation to look for sources in the image. Where DAOFind convolves the image with a perfect Gaussian whose FWHM has been specified by the user, this option convolves the image with the model PSF to identify all sources which most closely matches the PSF used. Those positions are then turned into a list that is fed to photutils DAOStarFinder code to measure them using the Gaussian models with a FWHM measured from the model PSF.

One benefit of this method is that features in the core of saturated or high S/N sources in the image that would normally be erroneously identified as a separate point-source by DAOFind will be recognized as part of the full PSF as far out as the model PSF extends.

For exposures which are comprised of images taken in different filters, the model PSF used is the drizzle combination of the model PSFs for each filter that comprised the image. This allows the code to best match the PSF found in the image of the `total detection` image. The model PSFs definitely do not exactly match the PSFs from the images due to focus changes and other telescope effects. However, they are close enough to allow for reasonably complete identification of actual point-sources in the images. Should the images suffer from extreme variations in the PSF, though, this algorithm will end up not identifying valid sources from the image. The user can provide their own library of PSFs to use in place of the model PSFs included with this package in order to more reliably match and measure the sources from their data. The user-provided PSFs can be used to directly replace the PSFs installed with this package as long as they maintain the same naming convention. All model PSFs installed with the code can be found in the `pars/psfs` directory, with all PSFs organized by instrument and detector. Each PSF file has a filename of `<instrument>_<detector>_<filter_name>.fits`. The model PSFs all extend at least 3.0" in radius in order to recognize the features of the diffraction spikes out as far as possible to avoid as many false detections as possible for saturated sources.

2.2: Aperture Photometry Measurement - Flux Determination

Aperture photometry is then preformed on the previously identified sources using a pair of concentric photometric apertures. The sizes of these apertures depend on the specific detector being used, and are listed below in table 1:

Table 5: Table 1: Aperture photometry aperture sizes

Instrument/Detector	Aper1 (arcsec)	Aper2 (arcsec)
ACS/HRC	0.03	0.125
ACS/SBC	0.07	0.125
ACS/WFC	0.05	0.15
WFC3/IR	0.15	0.45
WFC3/UVIS	0.05	0.15

Raw (non-background-subtracted) flux values are computed by summing up the enclosed flux within the two specified apertures using the photutils.aperture.aperture_photometry tool. Input values are detector-dependent, and can be found in the *_catalog_generation_all.json files described above in section 1.3.

Local background values are computed based on the 3-sigma-clipped mode of pixel values present in a circular annulus with an inner radius of 0.25 arcseconds and an outer radius of 0.50 arcseconds surrounding each identified source. This local background value is then subtracted from the raw inner and outer aperture flux values to compute the background-subtracted inner and outer aperture flux values found in the output .ecsv catalog file by the formula

$$f_{bgs} = f_{raw} - f_{bg} \cdot a$$

where

- f_{bgs} is the background-subtracted flux, in electrons per second
- f_{raw} is the raw, non-background-subtracted flux, in electrons per second
- f_{bg} is the per-pixel background flux, in electrons per second per pixel
- a is the area of the photometric aperture, in pixels

The overall standard deviation and mode values of pixels in the background annulus are also reported for each identified source in the output .ecsv catalog file in the "STDEV" and "MSKY" columns respectively (see Section 3 for more details).

2.3: Calculation of Photometric Errors

2.3.1: Calculation of Flux Uncertainties

For every identified source, the photutils.aperture_photometry() tool calculates standard deviation values for each aperture based on a 2-dimensional RMS array computed using the photutils.background.Background2d tool that we previously utilized to compute the 2-dimensional background array in order to background-subtract the detection image for source identification. We then compute the final flux errors as seen in the output .ecsv catalog file using the following formula:

$$\Delta f = \sqrt{\frac{\sigma^2}{g} + (a \cdot \sigma_{bg}^2) \cdot (1 + \frac{a}{n_{sky}})}$$

where

- Δf is the flux uncertainty, in electrons per second
- σ is the standard deviation of photometric aperture signal, in counts per second
- g is effective gain in electrons per count
- a is the photometric aperture area, in pixels
- σ_{bg} is standard deviation of the background
- n_{sky} is the sky annulus area, in pixels

2.3.2: Calculation of ABmag Uncertainties

Magnitude error calculation comes from computing $\frac{d(ABMAG)}{d(flux)}$. We use the following formula:

$$\Delta mag_{AB} = 1.0857 \cdot \frac{\Delta f}{f}$$

where

- Δmag_{AB} is the uncertainty in AB magnitude
- Δf is the flux uncertainty, in electrons per second
- f is the flux, in electrons per second

2.4: Calculation of Concentration Index (CI) Values and Flag Values

2.4.1: Calculation of Concentration Index (CI) Values

The Concentration index is a measure of the "sharpness" of a given source's PSF, and computed with the following formula:

$$CI = m_{inner} - m_{outer}$$

where

- CI is the concentration index, in AB magnitude
- m_{inner} is the inner aperture AB magnitude
- m_{outer} is the outer aperture AB magnitude

We use the concentration index to classify automatically each identified photometric source as either a point source (i.e. stars), an extended source (i.e. galaxies, nebulosity, etc.), or as an "anomalous" source (i.e. saturation, hot pixels, cosmic ray hits, etc.). This designation is described by the value in the "flags" column

2.4.2: Determination of Flag Values

The flag value associated with each source provides users with a means to distinguish between legitimate point sources, legitimate extended sources, and scientifically dubious sources (those likely impacted by low signal-to-noise ratio, detector artifacts, saturation, cosmic rays, etc.). The values in the "flags" column of the catalog are a sum of a one or more of these values. Specific flag values are defined below in table 2:

Table 6: Table 2: Flag definitions

Flag value	Meaning
0	Point source ($CI_{lower} < CI < CI_{upper}$)
1	Extended source ($CI > CI_{upper}$)
2	Bit value 2 not used in ACS or WFC3 sourcelists
4	Saturated Source
8	Faint Detection Limit
16	Hot pixels ($CI < CI_{lower}$)
32	False Detection: Swarm Around Saturated Source
64	False detection due proximity of source to image edge or other region with a low number of input images

Attention: The final output filter-specific sourcelists do not contain all detected sources. Sources that are considered scientifically dubious are filtered out and not written to the final source catalogs. For all detectors, sources with a flag value greater than 5 are filtered out. Users can adjust this value using a custom input parameter file and changing the "flag_trim_value" parameter. For more details on how to create a custom parameter file, please refer to the *generate_custom_svm_mvm_param_file* documentation page.

2.4.2.1: Assignment of Flag Values 0 (Point Source), 1 (Extended Source), and 16 (Hot Pixels)

Assignment of flag values 0 (point source), 1 (extended source), and 16 (hot pixels) are determined purely based on the concentration index (CI) value. The majority of commonly used filters for all ACS and WFC3 detectors have filter-specific CI threshold values that are automatically set at run-time. However, if filter-specific CI threshold values cannot be found, default instrument/detector-specific CI limits are used instead. Instrument/detector/filter combinations that do not have filter-specific CI threshold values are listed below in table 3 and the default CI values are listed below in table 4.

Table 7: Table 3: Instrument/detector/filter combinations that **do not** have filter-specific CI threshold values

Instrument/Detector	Filters without specifically defined CI limits
ACS/HRC	F344N
ACS/SBC	All ACS/SBC filters
ACS/WFC	F892N
WFC3/IR	None
WFC3/UVIS	None

Note: As photometry is not performed on observations that utilized grisms, prisms, polarizers, ramp filters,

or quad filters, these elements were omitted from the above list.

Table 8: Table 4: Default concentration index threshold values

Instrument/Detector	CI_{lower}	CI_{upper}
ACS/HRC	0.9	1.6
ACS/SBC	0.15	0.45
ACS/WFC	0.9	1.23
WFC3/IR	0.25	0.55
WFC3/UVIS	0.75	1.0

2.4.2.2: Assignment of Flag Value 4 (Saturated Source)

A flag value of 4 is assigned to sources that are saturated. The process of identifying saturated sources starts by first transforming the input image XY coordinates of all pixels flagged as saturated in the data quality arrays of each input flc/flt.fits images (the images drizzled together to produce the drizzle-combined filter image being used to measure photometry) from non-rectified, non-distortion-corrected coordinates to the rectified, distortion-corrected frame of reference of the filter-combined image. We then identify impacted sources by cross-matching this list of saturated pixel coordinates against the positions of sources in the newly created source catalog and assign flag values where necessary.

2.4.2.3: Assignment of Flag Value 8 (Faint Detection Limit)

A flag value of 8 is assigned to sources whose signal-to-noise ratio is below a predefined value. We define sources as being above the faint object limit if the following is true:

$$\Delta ABmag_{outer} \leq \frac{2.5}{snr \cdot log(10))}$$

Where

- $\Delta ABmag_{outer}$ is the outer aperture AB magnitude uncertainty
- snr is the signal-to-noise ratio, which is 1.5 for ACS/WFC and 5.0 for all other detectors.

2.4.2.4: Assignment of Flag Value 32 (False Detection: Swarm Around Saturated Source)

The source identification routine has been shown to identify false sources in regions near bright or saturated sources, and in image artifacts associated with bright or saturated sources, such as diffraction spikes, and in the pixels surrounding saturated PSF where the brightness level "plateaus" at saturation. We identify impacted sources by locating all sources within a predefined radius of a given source and checking if the brightness of each of these surrounding sources is less than a radially-dependent minimum brightness value defined by a pre-defined stepped encircled energy curve. The parameters used to determine assignment of this flag are instrument-dependent, can be found in the "swarm filter" section of the *_quality_control_all.json files in the path described above in section 1.3.

2.4.2.5: Assignment of Flag Value 64 (False Detection Due Proximity of Source to Image Edge or Other Region with a Low Number of Input Images)

Sources flagged with a value of 64 are flagged as "bad" because they are inside of or in close proximity to regions characterized by low or null input image contribution. These are areas where for some reason or another, very few or no input images contributed to the pixel value(s) in the drizzle-combined image. We identify sources impacted with this effect by creating a two-dimensional weight image that maps the number of contributing exposures for every pixel. We then check each source against this map to ensure that all sources and flag appropriately.

3: The Output Point Catalog File

3.1: Filename Format

Source positions and photometric information are written to a .ecsv (Enhanced Character Separated Values) file. The naming of this file is fully automatic and follows the following format: <TELESCOPE>_<PROPOSAL ID>_<OBSERVATION SET ID>_<INSTRUMENT>_<DETECTOR>_ <FILTER>_<DATASET NAME>_<CATALOG TYPE>.ecsv

So, for example if we have the following information:

- Telescope = HST
- Proposal ID = 98765
- Observation set ID = 43
- Instrument = acs
- Detector = wfc
- Filter name = f606w
- Dataset name = j65c43
- Catalog type = point-cat

The resulting auto-generated catalog filename will be:

- hst_98765_43_acs_wfc_f606w_j65c43_point-cat.ecsv

3.2: File Format and Comparison to the HLA Catalog

The .ecsv file format is quite flexible and allows for the storage of not only character-separated datasets, but also metadata. The first section (lines 4-17) contains a mapping that defines the datatype, units, and formatting information for each data table column. The second section (lines 19-27) contains information explaining STScI's use policy for HAP data in refereed publications. The third section (lines 28-48) contains relevant image metadata. This includes the following items:

- WCS (world coordinate system) name
- WCS (world coordinate system) type

- Proposal ID
- Image filename
- Target name
- Observation date
- Observation time
- Instrument
- Detector
- Target right ascension
- Target declination
- Orientation
- Aperture right ascension
- Aperture declination
- Aperture position angle
- Exposure start (MJD)
- Total exposure duration in seconds
- CCD Gain
- Filter name
- Total Number of sources in catalog

The next section (lines 50-66) contains important notes regarding the coordinate systems used, magnitude system used, apertures used, concentration index definition and flag value definitions:

- X, Y coordinates listed below use are zero-indexed (origin = 0,0)
- RA and Dec values in this table are in sky coordinates (i.e. coordinates at the epoch of observation and fit to GAIADR1 (2015.0) or GAIADR2 (2015.5)).
- Magnitude values in this table are in the ABMAG system.
- Inner aperture radius in pixels and arcseconds (based on detector platescale)
- Outer aperture radius in pixels and arcseconds (based on detector platescale)
- Concentration index (CI) formulaic definition
- Flag value definitions

Finally, the last section contains the catalog of source locations and photometry values. It should be noted that the specific columns and their ordering were deliberately chosen to facilitate a 1:1 exact mapping to the_daophot.txt catalogs produced by Hubble Legacy Archive. As this code was designed to be the HLA's replacement, we sought to minimize any issues caused by the transition. The column names are as follows (Note that this is the same left-to-right ordering in the .ecsv file as well):

- X-Center: 0-indexed X-coordinate position

- Y-Center: 0-indexed Y-coordinate position
- RA: Right ascension (sky coordinates), in degrees
- DEC: Declination (sky coordinates), in degrees
- ID: Object catalog index number
- MagAp1: Inner aperture brightness, in AB magnitude
- MagErrAp1: Inner aperture brightness uncertainty, in AB magnitude
- MagAp2: Outer aperture brightness, in AB magnitude
- MagErrAp2: Outer aperture brightness uncertainty, in AB magnitude
- MSkyAp2: Outer aperture background brightness, in AB magnitude
- StdevAp2: Standard deviation of the outer aperture background brightness, in AB magnitude
- FluxAp2: Outer aperture flux, in electrons/sec
- CI: Concentration index (MagAp1 – MagAp2), in AB magnitude
- Flags: See Section 2.4.2 for flag value definitions

3.3 Rejection of Cosmic-Ray Dominated Catalogs

Not all sets of observations contain multiple overlapping exposures in the same filter. This makes it impossible to ignore all cosmic-rays that have impacted those single exposures. The contributions of cosmic-rays often overwhelm any catalog generated from those single exposures making recognizing astronomical sources almost impossible amongst the noise of all the cosmic-rays. As a result, those catalogs can not be trusted. In an effort to only publish catalogs which provide the highest science value, criteria developed by the Hubble Legacy Archive (HLA) has been implemented to recognize those catalogs dominated by cosmic-rays and not provided as an output product.

Note: This rejection criteria is NOT applied to WFC3/IR or ACS/SBC data since they are not affected by cosmic-rays in the same way as the other detectors.

3.3.1 Single-image CR Rejection Algorithm

An algorithm has been implemented to identify and ignore cosmic-rays in single exposures. This algorithm has been used for ignoring cosmic-rays during the image alignment code used to determine the *a posteriori* alignment to GAIA.

This algorithm starts by evaluating the central moments of all sources from the segment catalog. Any source where the maximum central moment (as determined by photutils.segmentation.SourceProperties is 0 for both X and Y moments gets identified as cosmic-rays. This indicates that the source has a concentration of flux greater than a point-source and most probably represents a 'head-on cosmic-ray'.

In addition to these 'head-on cosmic-rays', 'glancing cosmic-rays' produce streaks across the detector. Those are identified by identifying sources with a minimum width (semiminor_axis) less than the FWHM of a point source and an elongation > 2. The width and elongation are also properties defined by photutils.segmentation.SourceProperties. The combination of these criteria allows for the identification of a vast majority of cosmic-rays. The DQ array of the single exposure then gets updated to flag those pixels identified as cosmic-rays based on these criteria. These DQ flags are then ONLY applied when creating the TotalProduct to limit the contribution of cosmic-rays from the total detection image. These flags are NOT used to generate any other product in order to avoid affecting the photometry or astrometry of any source from the total detection image any more than necessary.

3.3.2 Rejection Criteria

The rejection criteria has been defined so that if either the point source catalog or the segment catalog fails, then both catalogs are rejected and deleted.

In its simplest form the criteria for rejection is:

n_cat < thresh

where:

thresh = crfactor * (n1_residual * n1_exposure_time)**2 / texptime

and:

n_cat : Number of good point and extended sources in the catalog (flag < 2) crfactor : Number of expected cosmic-rays per second across the entire detector n1_exposure_time : amount of exposure time for all single filter exposures texptime : Total exposure time of the combined drizzle product n1_residual : Remaining fraction of cosmic-rays after applying single-image CR removal

The value of `crfactor` should be adjusted for sub-arrays to account for the smaller area being read out, but that logic has not yet been implemented. The values used in the processing of single-visit mosaics are:

segment-catalog crfactor : 300 point-catalog crfactor : 150

These numbers are deliberately set high to be conservative about which catalogs to keep. The CR rate varies with position in the orbit, and these are set high enough that it is rare for approved catalogs to be dominated by CRs (even though they can obviously have some CRs included.)

Finally, the `n1_residual` term gets set as a configuration parameter with a default value of 5% (0.05). This indicates that the single-image cosmic-ray identification process was expected to leave 5% of the cosmic-rays unflagged. This process can be affected by numerous factors, and having this as a user settable parameter allows the user to account for these effects when reprocessing the data manually. Pipeline processing, though, may still be subject to situations where this process does not do as well which can result in a catalog with a higher than expected contribution of cosmic-rays. Should this number of sources trigger the rejection criteria, these catalogs will be rejected and not written out.

Also note that we reject both the point and segment catalogs if either one fails this test. The reasoning behind that is that since the catalogs are based on the same image, it is unlikely that one catalog will be good and the other contaminated.

Should the catalogs fail this test, neither type of catalogs will be written out to disk for this visit.

4: Segmentation Catalog Generation

4.1: Source Identification with PhotUtils

For the segmentation algorithm the photutils.segmentation Astropy tool is used to identify sources in the background-subtracted multi-filter detection image. As is the case for the point-source detection algorithm, this is the juncture where the common background computed in Section 1.3, relevant for both the point and segment algorithms, is applied to the science data to begin the source detection process. To identify a signal as a source, the signal must have a minimum number of connected pixels, each of which is greater than its two-dimensional threshold image counterpart. Connectivity refers to how pixels are literally touching along their edges and corners, and the threshold image is the background RMS image (Section 1.3) multiplied by a configurable n-sigma value and modulated by a weighting scheme based upon the WHT extension of the detection image. Before applying the threshold, the detection image is filtered by the image kernel (Section 1.4) to smooth the data and enhance the ability to identify signal which is similar in shape to the kernel. This process generates a two-dimensional segmentation image or map where a segment is defined to be a number of connected pixels which are all identified by a numeric label and are considered part of the same source.

The segmentation map gets evaluated to determine the fraction of sources which are larger than a user-specified fraction of the image ("large" segments) and the total fraction of the image covered by segments. If either of these two scenarios is true, this is a strong indication the detection image is a crowded astronomical field. In such a crowded field, either the custom kernel or the Gaussian kernel (discussed in Section 1.4) can blend objects in close proximity together, making it difficult to differentiate between the independent objects. In extreme cases, a large number of astronomical objects are blended together and are mistakenly identified as a single segment covering a large percent of the image. To address this situation an alternative kernel is derived using the astropy.convolution.RickerWavelet2DKernel Astropy tool. The RickerWavelet2DKernel is approximately a Gaussian surrounded by a negative halo, and it is useful for peak or multi-scale detection. This new kernel is then used for the generation of an improved segmentation map from the multi-filter detection image.

The new segmentation map gets evaluated again to determine the number of "large" segments and the fraction of the image covered by segments. Should the new map indicate too many "large" segments or too much of the image covered by segments, then deblending gets applied to the map.

Because different sources in close proximity can be mis-identified as a single source, it is necessary to apply a deblending procedure to the segmentation map. The deblending is a combination of multi-thresholding, as is done by Source Extractor and the watershed technique.

> **Caution:** The deblending can be problematic if the background determination has not been well-determined, resulting in segments which are a large percentage of the map footprint. In this case, the deblending can take unreasonable amounts of time (e.g., days) to conclude. This led to the implementation of logic to **limit the use of deblending to only those segments which are larger than the PSF kernel**. This will result in some faint close sources being identified as a single source in the final catalog.

After deblending has successfully concluded, the resultant segmentation map is further evaluated based on an algorithm developed for the Hubble Legacy Archive to determine if big segments/blended regions persist or if a large percentage of the map is covered by segments.

The segmentation map derived from *and when used in conjunction with* the multi-filter detection image for

measuring source properties is **only** used to determine the centroids of sources.

Note: Questionable centroids (e.g., values of nan or infinity) and their corresponding segments are removed from the catalog entirely.

4.2: Isophotal Photometry Measurements

The actual isophotal photometry measurements are made on the single-filter drizzled images using the cleaned segmentation map derived from the multi-filter detection image. As was the case for the multi-filter detection image, the single-filter drizzled image is used in the determination of appropriate background and RMS images (Section 1.3). In preparation for the photometry measurements, the background-subtracted image, as well as the RMS image, are used to compute a total error array by combining a background-only error array with the Poisson noise of sources.

The isophotal photometry and morphological measurements are then performed on the background-subtracted single-filter drizzled image using the segmentation map derived from the multi-filter detection image, the background and total error images, the image kernel, and the known WCS with the photutils.segmentation.source_properties tool. The measurements made using this tool and retained for the output segment catalog are denoted in Table 5.

Table 9: Table 5: Isophotal Measurements - Subset of Segment Catalog Measurements and Descriptions

PhotUtils Variable	Catalog Column	Description
area	Area	Total unmasked area of the source segment (pixels^2)
background_at_centroid	Bck	Background measured at the centroid position
bbox_xmin	Xmin	Min X pixel in the minimal bounding box segment
bbox_ymin	Ymin	Min Y pixel in the minimal bounding box segment
bbox_xmax	Xmax	Max X pixel in the minimal bounding box segment
bbox_ymax	Ymax	Max Y pixel in the minimal bounding box segment
covar_sigx2	X2	Variance of position along X (pixels^2)
covar_sigxy	XY	Covariance of position between X and Y (pixels^2)
covar_sigy2	Y2	Variance of position along Y (pixels^2)
cxx	CXX	SExtractor's CXX ellipse parameter (pixel^-2)
cxy	CXY	SExtractor's CXY ellipse parameter (pixel^-2)
cyy	CYY	SExtractor's CYY ellipse parameter (pixel^-2)
elongation	Elongation	Ratio of the semi-major to the semi-minor length
ellipticity	Ellipticity	1 minus the Elongation
id	ID	Numeric label of the segment/Catalog ID number
orientation	Theta	Angle between the semi-major and NAXIS1 axes
sky_centroid_icrs	RA and DEC	Equatorial coordinates in degrees
source_sum	FluxIso	Sum of the unmasked data within the source segment
source_sum_err	FluxIsoErr	Uncertainty of FluxIso, propagated from input array
xcentroid	X-Centroid	X-coordinate of the centroid in the source segment
ycentroid	Y-Centroid	Y-coordinate of the centroid in the source segment

4.3: Aperture Photometry Measurements

The aperture photometry measurements included with the segmentation algorithm use the same configuration variable values and literally follow the same steps as what is done for the point algorithm as documented in Sections 2.2 - 2.4. The fundamental difference between the point and segment computations is the source position list used for the measurements.

5: The Output Segment Catalog Files

The metadata for the catalogs, both total detection and filter, as discussed in Sections 3.1 and 3.2, is pre-dominantly the same. The differences arise with respect to the specific columns present in the catalog. The naming convention for the catalogs is also the same except the filter name is replaced by the literal *total* for the total detection catalog: <TELESCOPE>_<PROPOSAL ID>_<OBSERVATION SET ID>_<INSTRUMENT>_<DETECTOR>_total_<DATASET NAME>_<CATALOG TYPE>.ecsv where CATALOG TYPE is either *point-cat* or *segment-cat*. Using the same example from Section 3.1, the resulting auto-generated segment total detection catalog filename will be:

- hst_98765_43_acs_wfc_total_j65c43_segment-cat.ecsv

and the filter catalog filename will be:

- hst_98765_43_acs_wfc_f606w_j65c43_segment-cat.ecsv

5.1: Total Detection Segment Catalog

The multi-filter detection level (aka total) catalog contains the fundamental position measurements of the detected source: ID, X-Centroid, Y-Centroid, RA, and DEC, supplemented by some of the aperture photometry measurements from *each* of the filter catalogs (ABMAG of the outer aperture, Concentration Index, and Flags). Effectively, the output Total Detection Segment Catalog is a distilled version of all of the Filter Segment Catalogs.

5.2: Filter Segment Catalog and Comparison to the HLA Catalog

Section 3.2 discusses the file format for the output filter catalogs, where the latter portion of this section is specific to the point catalogs. The general commentary is still relevant for the segment catalogs, except for the specific columns. In the case of the segment filter catalogs, the specific columns and the order of the columns were designed to be similar to the Source Extractor catalogs produced by the Hubble Legacy Archive (HLA) project.

Having said this, the PhotUtils/Segmentation tool is not as mature as Source Extractor, and it was not clear that all of the output columns in the HLA product were relevant for most users. As a result, some measurements in the HLA Source Extractor catalog may be missing from the output segment catalog at this time. The current Segment column measurements are as follows in Table 6 with the same left-to-right ordering as found in the .ecsv:

Table 10: Table 6: Segment Filter Catalog Measurements and Descriptions

Segment Column	SExtactor Column	Description	Units
X-Centroid	X_IMAGE	0-indexed Coordinate position	pixel
Y-Centroid	Y_IMAGE	0-indexed Coordinate position	pixel
RA	RA	Sky coordinate at epoch of observation	degrees
DEC	DEC	Sky coordinate at epoch of observation	degrees
ID		Catalog Object Identification Number	
CI	CI	Concentration Index	
Flags	FLAGS		
MagAp1	MAG_APER1	ABMAG of source, inner (smaller) aperture	ABMAG
MagErrAp1	MAGERR_APER1	Error of MagAp1	ABMAG
FluxAp1	FLUX_APER1	Flux of source, inner (smaller) aperture	electrons/s
FluxErrAp1	FLUXERR_APER1	Error of FluxAp1	electrons/s
MagAp2	MAG_APER2	ABMAG of source, outer (larger) aperture	ABMAG
MagErrAp2	MAGERR_APER2	Error of MagAp2	ABMAG
FluxAp2	FLUX_APER2	Flux of source, outer (larger) aperture	electrons/s
FluxErrAp2	FLUXERR_APER2	Error of FluxAp2	electrons/s
MSkyAp2		ABMAG of sky, outer (larger) aperture	ABMAG
Bck	BACKGROUND	Background, position of source centroid	electrons/s
Area		Total unmasked area of the source segment	pixels^2
MagIso	MAG_ISO	Magnitude corresponding to FluxIso	ABMAG
FluxIso	FLUX_ISO	Sum of unmasked data in source segment	electrons/s
FluxIsoErr	FLUXERR_ISO	Uncertainty, propagated from input error	electrons/s
Xmin	XMIN_IMAGE	Min X pixel in minimal bounding box segment	pixels
Ymin	YMIN_IMAGE	Min Y pixel in minimal bounding box segment	pixels
Xmax	XMAX_IMAGE	Max X pixel in minimal bounding box segment	pixels
Ymax	YMAX_IMAGE	Max Y pixel in minimal bounding box segment	pixels
X2	X2_IMAGE	Variance along X	pixel^2
Y2	Y2_IMAGE	Variance along Y	pixel^2
XY	XY_IMAGE	Covariance of position between X and Y	pixel^2
CXX	CXX_IMAGE	SExtractor's ellipse parameter	pixel^2
CYY	CYY_IMAGE	SExtractor's ellipse parameter	pixel^2
CXY	CXY_IMAGE	SExtractor's ellipse parameter	pixel^2
Elongation	ELONGATION	Ratio of semi-major to semi-minor length	
Ellipticity	ELLIPTICITY	The value of 1 minus the elongation	
Theta	THETA_IMAGE	Angle between semi-major and NAXIS1 axes	radians

6: Reading The Output Catalog Files

All of the Point and Segmentation catalogs, filter and total, are Enhanced Character-Separated Values (ECSV) files which are human-readable ASCII tables. As such, it is straight-foward to access the astronomical source data contained in the rows of the files in a programmatic way via Astropy or Pandas.

An Astropy example with a Segmentation filter catalog will generate the following Astropy table (abridged view):

```
>>> from astropy.table import Table
>>> astro_tab=Table.read("hst_15064_11_acs_wfc_f814w_jdjb11_segment-cat.ecsv",␣
↪format="ascii.ecsv")
>>> astro_tab
<Table length=375>
X-Centroid Y-Centroid       RA           DEC         ID      CI    ...    CYY    ␣
↪  CXY    Elongation Ellipticity  Theta
   pix        pix          deg           deg              mag(AB) ...  1 / pix2 ␣
↪1 / pix2                          rad
 float64    float64      float64       float64     int64  float64 ...  float64  ␣
↪float64   float64     float64   float64
---------- ---------- ------------ ------------ ------- ------- ... --------- -
↪-------- ---------- ----------- --------
  3774.045     87.935   313.5799763    -0.1839533       1   2.144 ...   0.25651 ␣
↪-0.13623       1.30        0.23   52.349
  3630.189    101.246   313.5819743    -0.1837685       2   1.642 ...   0.12165 ␣
↪-0.00195       1.03        0.03   82.412
```

The "comment" parameter in this Pandas example is necessary so that the reader will skip over the header lines which it cannot parse. The first line which is actually read is the "line 0" (header=0) which consists of the ascii column names. The result is a Pandas dataframe for this example of the Point filter catalog:

```
>>> import pandas
>>> df=pandas.read_csv("hst_15064_11_acs_wfc_f814w_jdjb11_point-cat.ecsv", sep="
↪", header=0, comment="#")
>>> df
        X-Center     Y-Center          RA       DEC   ID  ...   MSkyAp2 ␣
↪StdevAp2      FluxAp2        CI  Flags
0    3774.738972    89.759486  313.579967 -0.183928    1  ...  0.165745  0.
↪009700     4.732320  1.561092      1
1    3630.522602   102.347181  313.581970 -0.183753    2  ...  0.151377  0.
↪227345   834.948972  1.189462      4
```

3.2.3 Multi-Visit Mosaics

Multi-visit Mosaic Processing

Multi-Visit Mosaic (MVM)
: A Multi-Visit Mosaic (MVM) is a single image (product) made by combining all observations taken of the same part of the sky.

Observations taken of the same part of the sky over all the years that HST has been operational can enable unique science due to the high-resolution of the HST cameras. Generating useful mosaics from these observations, though, requires solving a number of key problems; namely,

- aligning all the images to the same coordinate system
- define the size on the sky of each mosaic
- defining what exposures should go into each mosaic

MVM processing implemented as part of the Hubble Advanced Products (HAP) pipeline generates new products based on solutions implemented for these critical issues.

Alignment

Generating MVM products relies entirely on the definition of the WCS of each input image in order to define where each exposure will land in the output mosaic. Errors in an input exposure's WCS will lead to misalignment of overlapping exposures resulting in smeared sources or even multiple images of the overlapping sources in the output mosaic. Errors in the WCS primarily stem from images being aligned to different astrometric catalogs. For example, one exposure may be aligned successfully to the GAIADR2 catalog, yet another overlapping exposure can only be aligned to the 2MASS catalog due to the lack of GAIA sources in the offset exposure.

MVM processing relies entirely on the alignment that can be performed on the exposures during standard pipeline processing and subsequently as part of the Single-Visit Mosaic (SVM) processing. This allows each visit to be aligned to the most accurate catalog available while taking into account the proper motions of the astrometric sources in the field based on the date the exposures were taken in the visit.

By default, the most current GAIA catalog will be used as the default astrometric catalog for aligning all exposures which typically results in images that are aligned well enough to avoid serious misalignment artifacts in the regions of overlap. Unfortunately, due to the nature of some of the fields observed by HST, not all exposures can be aligned to a GAIA-based catalog using GAIA astrometric sources. As a result, the sources used for alignment will have much larger errors on the sky compared to the errors of GAIA astrometric sources which can lead to offsets from the GAIA coordinate system of a significant fraction of an HST pixel (or even multiple HST pixels). This will lead to artifacts in MVM products where such exposures overlap other exposures aligned to other catalogs, especially GAIA-based catalogs.

Defining the Mosaic on the Sky

MVM products need to be defined in order to understand what input exposures will contribute to each mosaic. The solution implemented for HAP MVM products relies on tesselation of the entire sky using the same basic tiles defined by the PanSTARRS project as described at PanSTARRS Sky tessellation patterns.

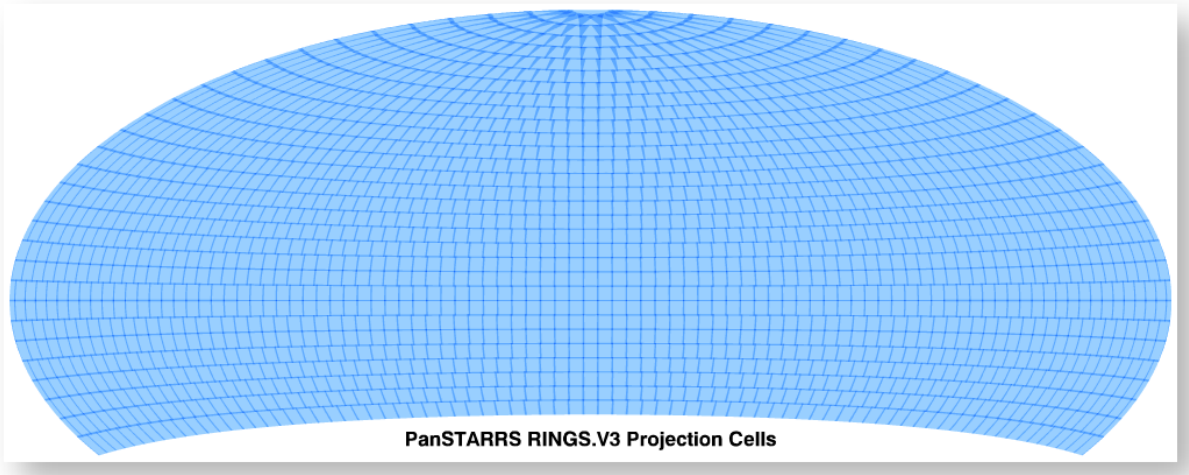


Fig. 1: Aitoff plot of all 2,009 PS1 projection cells for the 3PI survey. The coverage extends from declination 30° to the north celestial pole.

MVM processing uses these pre-defined projection cells after extending them to cover the entire sky with cells that are actually 4.2°x4.2° in size and are on rings spaced by 4° apart in declination. This provides sufficient overlap to allow data from one projection cell to be merged with data from a neighboring cell if necessary. These cells, defined as the ProjectionCell object in the code and referred to as **ProjectionCell**, are indexed starting at 0 at = 90° (South Pole) and ending at 2643 at = +90° (North Pole). As stated in the PanSTARRS page:

> The projection cell centers are located on lines of constant declination spaced 4° apart. At a given declination, the pointing centers are equally spaced in right ascension around the sky, with the number of RA points changing to account for the convergence of RA lines in the spherical sky. The pattern is defined to cover the entire sky from = 90° to +90°… Within a given declination zone, the projection cells are centered at RA(n) = n = n 360°/M where M is the number of RA cells in the zone. Finally, the projection cells themselves are numbered consecutively (ordered by increasing RA) starting at 0 at the south pole, 1–9 at = 86°, etc.

All these definitions, including how finely each ProjectionCell should be divided to define the SkyCells, are encoded in a FITS table installed as part of the `drizzlepac` package:

> drizzlepac/pars/allsky_cells.fits

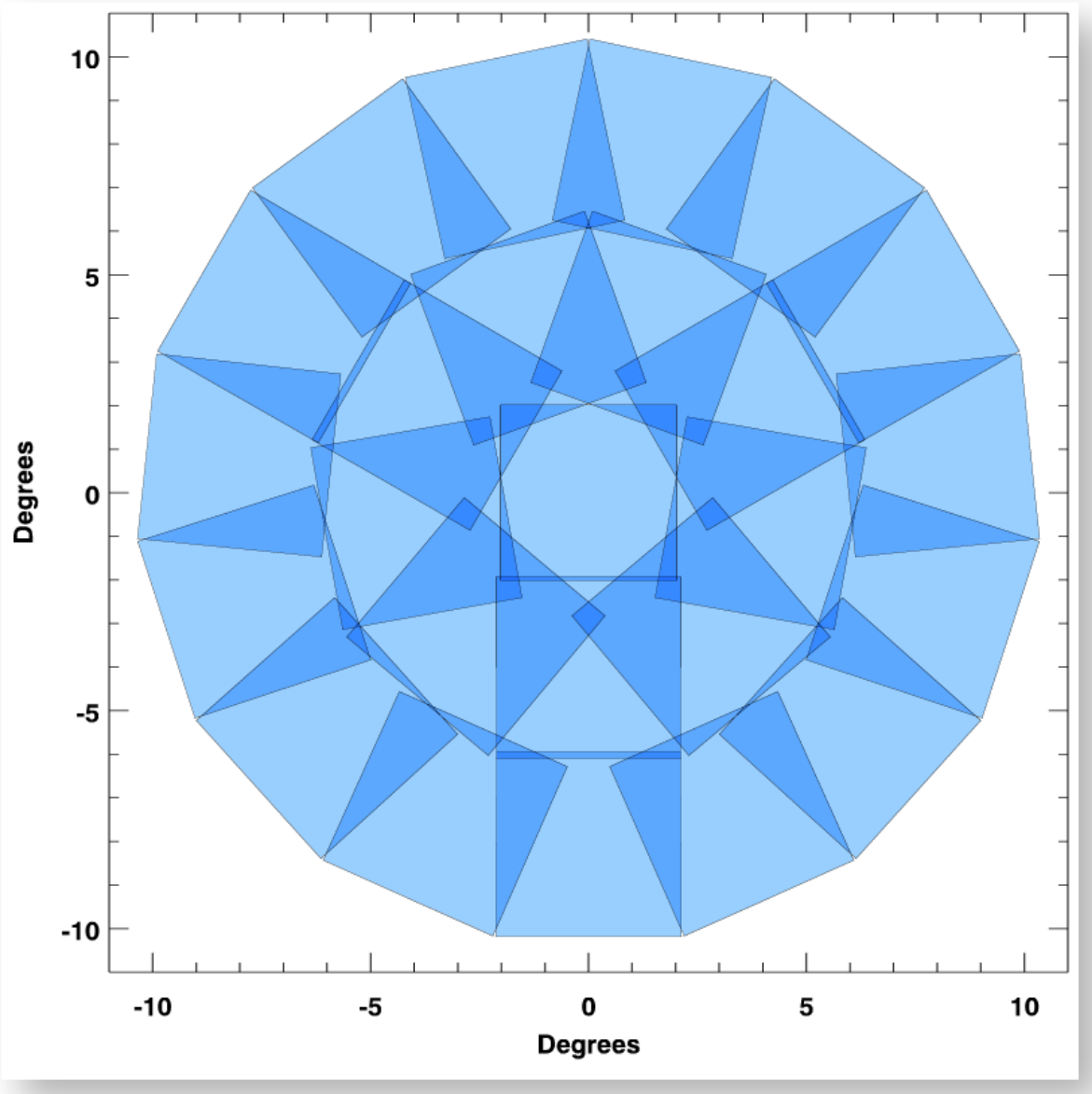


Fig. 2: PS1 projection cells near the north celestial pole, where the image overlap is greatest due to convergence of the RA grid. The projection cells are 4°x4° in size and are on rings spaced by 4° in declination.

Defining each SkyCell

Each ProjectionCell is split into a grid of 21 x 21 'sky cells' which serves as the most basic MVM product generated during MVM processing. Sky cells, as defined as the SkyCell object in the code and referred to as **SkyCell**, are approximately 0.2° x 0.2° in size (~18000 x ~18000 pixels) and they have the same WCS as the ProjectionCell. Each SkyCell gets identified by its position within the ProjectionCell as shown in this figure:

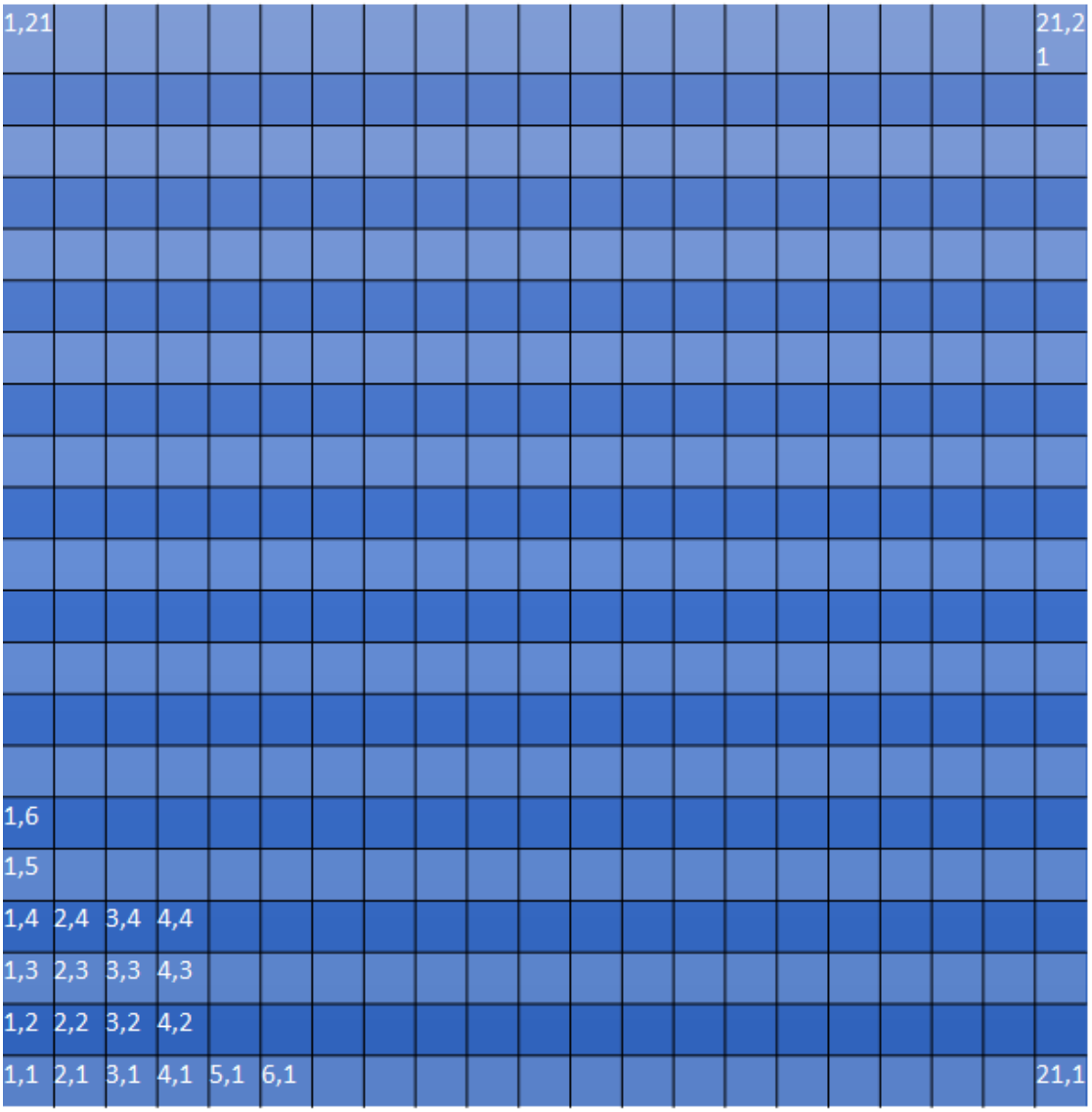


Fig. 3: Numbering convention for SkyCells within a Projection Cell used for naming the SkyCell.

The numbering of the SkyCells within a ProjectionCell starts in the lower left corner at (1,1) corresponding to the cell with the lowest declination and largest RA since the SkyCell is oriented so that the Y axis follows a line of RA pointing towards the North Pole. This indexing provides a way to identify uniquely any position on the sky and can be used as the basis for a unique filename for all products generated from the exposures that overlap each SkyCell. Mosaics generated for each SkyCell uses this indexing to create files with names using the convention:

skycell-p<PPPP>x<XX>y<YY>

where:

Element	Definition
<PPPP>	ProjectionCell ID as a zero-padded 4 digit integer
<XX>,<YY>	SkyCell ID within ProjectionCell as zero-padded 2 digit integers

The WCS for each SkyCell gets defined as a subarray of the Projection cell's WCS. This allows data across SkyCells in the same ProjectionCell to be combined into larger mosaics as part of the same tangent plane without performing any additional resampling.

Code for Defining SkyCell ID

The code for interfacing with the cell definitions table can be imported in Python using:

```
from drizzlepac.haputils import cell_utils
```

Determining what SkyCells overlap any given set of exposures on the sky can be done using the function:

```
sky_cells_dict = cell_utils.get_sky_cells(visit_input, input_path=None)
```

where **visit_input** is the Python list of filenames of FLT/FLC exposures. These files can be any set of FLT/FLC files and the WCS solutions defined in them will, by default, **be used as-is** to create the final combined mosaics. The MVM products generated in the HST pipeline and stored in the HST archive will be generated using FLT/FLC files that have been aligned to an astrometric catalog like GAIAeDR3 during SVM processing, if alignment was possible at all for the exposure. The full set of parameters that can be used to control the sky cell definitions and IDs can be found in the *MVM Processing Code API page*.

Exposures in an input list are assumed to be in the current working directory when running the code, unless **input_path** has been provided which points to the location of the exposures to be processed. The return value **sky_cells_dict** is a dictionary where the keys are the names (labels) of each overlapping SkyCell and the value is the actual SkyCell object which contains the footprint and WCS (among other details) of the SkyCell.

For example, observations from HST proposal 14175 were taken to study NGC 4594 (Sombrero Galaxy) using both the ACS and WFC3 cameras. The footprints of all the HST/ACS and HST/WFC3 observations taken in this part of the sky as shown by MAST can be seen here:

There were 25 FLT and FLC exposures taken as part of this proposal making up these footprints. The list of these files defined what was provided as **visit_input** to the *get_sky_cells()* function to get these SkyCell definitions:

```
sky_cells_dict = cell_utils.get_sky_cells(visit_input, input_path=None)
sky_cells_dict

{'skycell-p1889x07y19': SkyCell object: skycell-p1889x07y19,
```

(continues on next page)

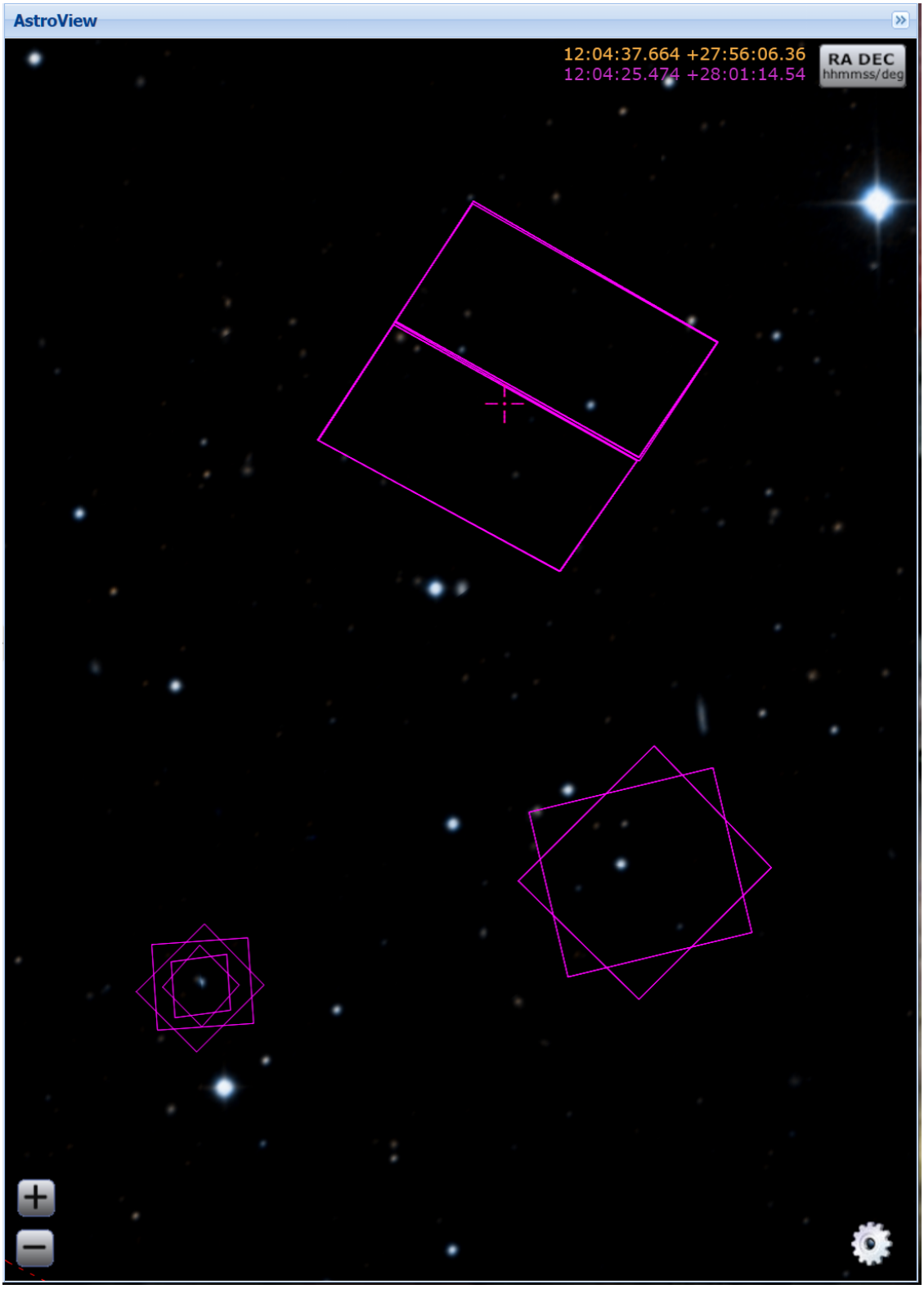


Fig. 4: Footprints of HST/ACS and HST/WFC3 observations of the the quasar PG1202+281 and surrounding area.

(continued from previous page)

```
 'skycell-p1889x07y20': SkyCell object: skycell-p1889x07y20,
 'skycell-p1970x15y03': SkyCell object: skycell-p1970x15y03,
 'skycell-p1970x15y02': SkyCell object: skycell-p1970x15y02,
 'skycell-p1970x16y02': SkyCell object: skycell-p1970x16y02}
```

This indicates that these exposures overlap 5 SkyCells in 2 ProjectionCells **p1889** and **p1970** with the WCS defined in the defined SkyCell object for each SkyCell entry in this dictionary. Each SkyCell object includes a list of all the exposures that overlap just that SkyCell, which can be used to generate those mosaics. Full details of the contents of the SkyCell object can be found in the *Multi-visit Processing API documentation*.

We can see that one SkyCell includes all exposures using:

```
for sky_cell in sky_cells_dict:
    print(sky_cell, len(sky_cells_dict[sky_cell].members))

skycell-p1889x07y19 25
skycell-p1889x07y20 8
skycell-p1970x15y03 8
skycell-p1970x15y02 17
skycell-p1970x16y02 8
```

All subsequent examples will use the exposures for SkyCell **skycell-p1889x07y19**.

Input Poller File

The MVM processing could simply combine whatever input files are present in the current working directory. However, that may result in working with more than 1 SkyCell at a time which can, for some steps, end up requiring more memory or disk space than is available on the system. Therefore, the code relies on an input `poller file` which specifies exactly what files should be processed at one time. This ASCII CSV-formatted input `poller file` will only contain the names of exposures which overlap only a single SkyCell regardless of instrument, detector or any other observational configuration.

Generating one of these input 'poller' files with the filename *skycell-p1889x07y19_mvm_input.txt* can be done using the same SkyCell dictionary defined earlier with the commands:

```
from drizzlepac.haputils import make_poller_files as mpf

with open('skycell-p1889x07y19_input.txt', 'w') as fout:
    _ = [fout.write(f'{sky_cell}\n') for sky_cell in sky_cells_dict['skycell-
↪p1889x07y19'].members]

mpf.generate_poller_file('skycell-p1889x07y19_input.txt',
                         poller_file_type='mvm',
                         output_poller_filename='skycell-p1889x07y19_mvm_poller.
↪txt',
                         skycell_name='skycell-p1889x07y19')
```

The full description of the function used to create this poller file can be found in the *Multi-visit Processing Code API documentation*.

This input file *skycell-p1889x07y19_mvm_input.txt* can then be used as input to the top-level MVM processing code using:

```
from drizzlepac import hapmultisequencer
rv = hapmultisequencer.run_mvm_processing("skycell-p1889x07y19_mvm_poller.txt")
```

The `poller file` contains 1 line for each input exposure for a given SkyCell. The form of the file, though, is a comma-separated (CSV) formatted file with all the same information as the SVM input files plus a couple of extra columns; namely,

- SkyCell ID
- status of MVM processing

An example of an exposure's line in the poller file would be:

```
hst_12286_0r_acs_wfc_f775w_jbl70rtv_flc.fits,12286,BL7,0R,486.0,F775W,WFC,
↪skycell-p1889x07y19,NEW,g:\data\mvm\p1889x07y19\hst_12286_0r_acs_wfc_f775w_
↪jbl70rtv_flc.fits
```

where the elements of each line are defined as:

```
filename, proposal_id, program_id, obset_id, exptime, filters, detector, skycell-
↪p<PPPP>x<XX>y<YY>, [OLD|NEW], pathname
```

The SkyCell ID will be included in this input information to allow for grouping of exposures into the same SkyCell layer based on filter, exptime, and year.

The value of **'NEW'** specifies that this exposure should be considered as never having been combined into this SkyCell's mosaic before. A value of **'OLD'** instead allows the code to recognize layers that are unaffected by 'NEW' data so that those layers can be left alone and NOT processed again unnecessarily. As such, it can serve as a useful summary of all the input exposures used to generate the mosaics for the SkyCell.

Defining SkyCell Layers

Defining the SkyCell for a region on the sky allows for the identification of all exposures that overlap that WCS. However, creating a single mosaic from data taken with different detectors and filters would not result in a meaningful result. Therefore, the exposures that overlap each SkyCell get grouped based on the detector and filter used to take the exposure to define a 'layer' of the SkyCell. Each layer can then be generated as the primary basic image product for each SkyCell. Exposures taken with spectroscopic elements, like grisms and prisms, and exposures taken of moving targets can not be used to create layers due to the inability to align them with the rest of the observations. Therefore, only images taken with standard filters (like the WFC3/UVIS F275W filter) will be used to define SkyCell mosaics (layers).

The default plate scale for all MVM image products for each SkyCell has been defined as 0.04"/pixel to match the higher resolution imaging performed by the WFC3/UVIS detector. However, WFC3/IR data suffers from serious resampling artifacts when drizzling IR data to that plate scale. So in addition to creating IR mosaics

at the 0.04”/pixel ‘fine’ plate scale, IR mosaics are also generated at a ‘coarse’ plate scale of 0.12”/pixel to minimize the resampling artifacts while also being easily scaled to the ‘fine’ plate scale mosaics.

SkyCell Layers Example

For example, observations were taken with Proposals 12286 and 12903 using both the ACS and WFC3 cameras and multiple filters. The ACS observations were taken with the ACS/WFC detector using the F775W and F850LP filters, while the WFC3 observations were taken using the IR detector using the F105W, F125W and F160W filters as well as the UVIS detector using the F475W. All these observations fall within the SkyCell at position **x07y19** in the ProjectionCell **p1889**, but given the dramatic plate scale differences, these observations can not be used to create a single mosaic.

The different observing modes used for observations in this SkyCell end up being organized as 9 separate layers (mosaics); namely,

- wfc3_uvis_f475w (0.04”/pixel)
- wfc3_ir_f105w_coarse (0.12”/pixel)
- wfc3_ir_f105w (0.04”/pixel)
- wfc3_ir_f125w_coarse (0.12”/pixel)
- wfc3_ir_f125w (0.04”/pixel)
- wfc3_ir_f160w_coarse (0.12”/pixel)
- wfc3_ir_f160w (0.04”/pixel)
- acs_wfc_f850lp (0.04”/pixel)
- acs_wfc_f775w (0.04”/pixel)

Since they all have the same WCS, modulo the plate scale differences, they can be overlaid pixel-by-pixel with each other for analysis.

You can verify this interactively by directly calling the code that interprets the input ‘poller’ file using:

```
from drizzlepac.haputils import poller_utils

obs_dict, tdp_list = poller_utils.interpret_mvm_input('skycell-p1889x07y19_mvm_
↪poller.txt',
                                                      log_level=poller_utils.
↪logutil.logging.INFO)
for layer in obs_dict:
    print(obs_dict[layer]['info'])

skycell-p1889x07y19 wfc3 uvis f475w all all 1 drz fine
skycell-p1889x07y19 wfc3 ir f105w all all 1 drz coarse
skycell-p1889x07y19 wfc3 ir f105w all all 1 drz fine
skycell-p1889x07y19 wfc3 ir f125w all all 1 drz coarse
skycell-p1889x07y19 wfc3 ir f125w all all 1 drz fine
```

(continues on next page)

(continued from previous page)

```
skycell-p1889x07y19 wfc3 ir f160w all all 1 drz coarse
skycell-p1889x07y19 wfc3 ir f160w all all 1 drz fine
skycell-p1889x07y19 acs wfc f850lp all all 1 drc fine
skycell-p1889x07y19 acs wfc f775w all all 1 drc fine

print(tdp_list)

[<drizzlepac.haputils.product.SkyCellProduct at 0x1fd04dcc220>,
 <drizzlepac.haputils.product.SkyCellProduct at 0x1fd04d49a60>,
 <drizzlepac.haputils.product.SkyCellProduct at 0x1fd04d49cd0>,
 <drizzlepac.haputils.product.SkyCellProduct at 0x1fd04dccd30>,
 <drizzlepac.haputils.product.SkyCellProduct at 0x1fd04dcc0a0>,
 <drizzlepac.haputils.product.SkyCellProduct at 0x1fd04dccfd0>,
 <drizzlepac.haputils.product.SkyCellProduct at 0x1fd04dcca60>,
 <drizzlepac.haputils.product.SkyCellProduct at 0x1fd117a1e50>,
 <drizzlepac.haputils.product.SkyCellProduct at 0x1fd04d53910>]
```

These objects define the WCS, inputs and filenames for each layer for use in creating these products. Full details of the contents of the **SkyCellProduct** can be found in *Multi-visit Processing Code API documentation*.

MVM Processing Steps

The definitions for the ProjectionCell and SkyCell allow for all HST observations to be processed into a logical set of image mosaic products regardless of how many observations cover any particular spot on the sky while tying them all together in the same astrometric reference frame (as much as possible, anyway). The steps taken to generate these MVM products can be summarized as:

- Determine what SkyCell or set of SkyCells each exposure overlaps
- Copy all relevant exposures for a given SkyCell into a single directory for that SkyCell
- Rename input exposures to have MVM-specific filenames
- Generate input file to be used for processing each SkyCell
- Evaluate all input exposures to define all layers needed for the SkyCell
- Determine which layer to process
- Drizzle all exposures for each layer to be processed to create new mosaic product for that layer

Running MVM Processing

The primary function used to perform MVM processing interactively in a Python session is:

```
from drizzlepac import hapmultisequencer
r = hapmultisequencer.run_mvm_processing(input_filename)
```

The function takes several optional parameters to control aspects of the processing, with full details of this function's parameters and all related functions being described in *Multi-visit Processing API*.

SkyCell Membership

Data from HST gets organized based on a filename derived from the proposal used to define the observations and how HST should take them. MVM processing, on the other hand, focuses on how the observations relate to each other on the sky based on the WCS information. The `cell_utils` module includes the code used to interpret the WCS information for exposures and determine what ProjectionCells and SkyCells each exposure overlaps as shown in the section on the *Code for Defining SkyCell ID*.

This code can be called on any set of user-defined exposures to determine for the first time what SkyCells the exposures overlap. During HST pipeline processing, this code gets called for the exposures from each visit after they have finished SVM processing and after they have been aligned as much as possible to the latest astrometric reference frame (such as the GAIAeDR3 catalog). Using these updated WCS solutions provides the most accurate placement of the exposures on the sky and therefore in the correct SkyCell.

Copy Data

The MVM processing code requires the input exposures to be located in the current working directory as the processing may require the ability to update the input files with the results of the MVM processing. Each input file may also overlap more than 1 SkyCell. As a result, the input files get copied into a directory set up specifically for processing a given SkyCell. This results in each input file being copied into as many directories as needed to support creating the mosaics for as many SkyCells as desired while protecting the integrity of the original input files and their WCS solutions.

Rename Input Files

The MVM processing code works on the input files provided using the WCS solutions defined in the headers of the input files as-is for the initial implementation. However, in order to preserve the solutions defined by previous processing steps, these files are renamed based on the SkyCell to be generated in the current working directory.

For reference, the original pipeline-assigned name has the format of:

`IPPPSSOOT_flc.fits`

such as

- jcz906dvq_flc.fits

- icz901wpq_flc.fits

The MVM filename defined for the **input exposures** follows the convention.

```
hst_skycell-p<PPPP>x<XX>y<YY>_<instr>_<detector>_<filter>_<ipppssoo>_fl[ct].
fits
```

where:

Element	Definition
<PPPP>	ProjectionCell ID as a zero-padded 4 digit integer
<XX>,<YY>	SkyCell ID within ProjectionCell as zero-padded 2 digit integers
<instr>	Name of HST instrument from **INSTRUME** header keyword
<detector>	Name of detector from **DETECTOR** header keyword
<filter>	Name of filter from **FILTER** or **FILTER1,FILTER2** header keyword(s)
<ipppssoo>	Pipeline-assigned **IPPPSSOO** designation from original input filename
fl[ct]	Suffix of either **flt** or **flc** from original input filename

This insures that each exposure gets renamed in a way that allows them to be easily identified with respect to the **output SkyCell layer** the exposure contributes to during MVM processing.

Note: The currently implemented MVM processing **does not update the WCS and DQ arrays** of these input files in any way. As a result, they only get used as intermediate products, and get deleted automatically upon **successful completion** of MVM processing. Should future updates to MVM processing be implemented, for example to further refine the alignment, then these products would get updated at that time and be added as a new product to the HST archive instead of being deleted.

Primary MVM Processing Interface

MVM processing gets controlled through a single function:

```
from drizzlepac import runmultihap

rv = runmultihap.process(input_filename)
```

This function takes as either form of the input file generated for the input exposures in the current directory as the input parameter `input_filename`. This function then performs all the processing steps automatically to generate the image mosaics from the exposures listed in the input file.

There are times, though, when the default processing needs to be revised to account for the science goals of the processing or to account for the exposures available as inputs. The following environmental variables can be used to control how the MVM processing deals with various types of input files:

MVM_INCLUDE_SMALL
: This controls whether or not layers are created for ACS/HRC or ACS/SBC exposures given their small field-of-view. By default, this is turned on ('true') so that these layers are created.

MVM_ONLY_CTE
: This controls whether or not to include exposures which have NOT been CTE-corrected due to the potential impact to the output mosaics PSFs from including exposures with CTE tails. By default, this is actually turned off ('false') so that all data gets used.

These variables can be set to values of 'on', 'off', 'true', 'false', 'yes' or 'no' in the operating system environment or even in the python environment using `os.environ`.

Additionally, the function *run_mvm_processing()*, which gets called by *perform()*, has the ability to enable an additional attempt to align all the input exposures to the latest astrometric catalog, as well as limit the size of the output mosaics. Full details of these parameters are available in the discussion of the *Multi-visit Processing Code API documentation*.

Define SkyCell Layers

SkyCells define the WCS that will be used for all the observations for any given region on the sky. However, it doesn't make sense to create a single image from all the exposures due to differences in the detectors, pixel sizes, PSFs and filters used for the observations. It can even be argued that observations taken too far apart in time should also not be combined, or observations taken with dramatic differences in exposure time should not be combined. As a result, the concept of a SkyCell 'layer' was implemented to organize all the exposures for a SkyCell into sets of exposures which can be combined to create useful, and hopefully scientifically interesting, mosaics.

The most basic definition for a layer organizes the exposures based on the following criteria:

- instrument
- detector
- filter

Using the renamed input files, the MVM processing code organizes all the exposures based on these criteria to identify what layers could be generated from all the inputs. For example, a SkyCell with ACS/WFC3 F814W exposures and WFC3/UVIS F606W exposures would result in 2 SkyCell layers being defined; namely, one for each set of exposures.

Note: Observations taken with a spectroscopic element, like grisms or prisms, will not be used to define SkyCell layers.

Determine Layers to Process

Specifying what input exposures need to be used to create a SkyCell layer mosaic serves as a critical feature of the input file. Input files simply listing filenames indicate that ALL exposures specified should be used to create MVM products. However, the more descriptive `poller file` CSV format input file provides the ability to limit the processing to only new exposures, while treating already archived versions of the MVM products for all the other SkyCell layers as fully updated and not in need of any further processing. This control comes from the entry that specifies 'NEW' or 'OLD', with only those layers with at least 1 'NEW' entry getting defined for processing.

Create SkyCell Mosaics

The input files get interpreted to define the image products that need to be created when combining the exposures. These mosaics get generated by drizzling the input exposures onto the WCS defined for the SkyCell, creating a separate mosaic for each layer from all the exposures with the same filter/detector/instrument configuration.

The drizzle parameters used to create these products are determined based on the average number of exposures for all the exposed pixels in the SkyCell layer. This only serves as an approximation of what would work best across the entire SkyCell layer, as some portions may only have a single exposure while other regions may have many overlapping exposures. However, this still works reasonably well due to the fact that only the following drizzle steps are actually applied when creating the MVM products:

- sky matching
- final drizzling with bad pixel rejection

These products, thus, rely entirely on the DQ arrays to be updated by SVM and pipeline processing to flag cosmic-rays and detector artifacts as bad pixels so those pixels get rejected when creating the MVM product. In addition, they rely on the WCS solutions provided by previous processing as well to place the exposures in the final MVM product.

The parameters used for creating these drizzle products can be found installed with the package's code. You can find the files using:

```
import os
import drizzlepac
#
# Directory is:  drizzlepac/pars/hap_pars/mvm_parameters
#
os.chdir(os.path.join(drizzlepac.__path__, 'pars', 'hap_pars', 'mvm_parameters'))
```

There are separate configuration files for each detector based on the number of average exposures in the output frame.

The final output products get created with a final output array size that is trimmed down to only the subsection of the entire SkyCell which has HST data in any layer.

Finally, the output drizzle product will have a filename that follows the basic convention used to rename the input exposures without the final 'ipppssoo' designation; namely,

```
hst_skycell-p<PPPP>x<XX>y<YY>_<instr>_<detector>_<filter>_<layer>_dr[cz].
fits
```

where:

Element	Definition
<PPPP>	ProjectionCell ID as a zero-padded 4 digit integer
<XX>,<YY>	SkyCell ID within ProjectionCell as zero-padded 2 digit integers
<instr>	Name of HST instrument from **INSTRUME** header keyword
<detector>	Name of detector from **DETECTOR** header keyword
<filter>	Name of filter from **FILTER** or **FILTER1,FILTER2** header keyword(s)
<layer>	Layer-specific designation: coarse-all or all
dr[cz]	Suffix of either **DRZ** or **DRC** based on input filenames

The <*layer*> component of the MVM filename indicates the plate scale and any other criteria used to define the layer such as exposure time or date range of exposures used to create the layer. At present, only WFC3/IR data gets generated with both the default (fine) plate-scale of 0.04"/pixel as well as the IR-native "coarse" plate scale which show up with a <*layer*> term of **coarse-all**. Initial processing does not apply any additional definitions for the layers, and thus the remainder of the initially generated MVM products simply have a <*layer*> term of **all**. Future processing may enable generation of additional layers based on date ranges for SkyCells which have massive amount of exposures over a large range of dates, in which case this <*layer*> term will be updated to reflect those ranges. Additionally, the code can be run interactively to enable generation of additional layers based on exposure time ranges as well. See explanation of the processing code functions for more details.

Multi-visit Mosaic Products

Multi-Visit Mosaic (MVM)
: A Multi-Visit Mosaic (MVM) is a single image (product) made by combining all observations taken of the same part of the sky.

Observations taken of the same part of the sky over all the years that HST has been operational can enable unique science due to the high-resolution of the HST cameras. Generating useful mosaics from these observations, though, requires solving a number of key problems; namely,

- aligning all the images to the same coordinate system
- define the size on the sky of each mosaic
- defining what exposures should go into each mosaic

MVM processing implemented as part of the Hubble Advanced Products (HAP) pipeline generates new products based on solutions implemented for these critical issues. These new products, SkyCell layers, are unlike other HST images due to the fact that they consist of many exposures taken at different times, sometimes years apart. The format of these products and the new aspects of these products are described in the following sections.

SkyCell Layers

The most basic MVM product would be the SkyCell layer as described in *Defining SkyCell Layers* section. These layers represent all the exposures taken in a given detector/filter combination in that SkyCell (position on the sky). As a example, the exposures for sky cell **p1889x07y19** define 9 separate layers; namely,

Layer Name	Plate scale	SkyCell filename
wfc3_uvis_f475w	0.04”/pixel	hst_skycell-p1889x07y19_wfc3_uvis_f475w_all_drz.fits
wfc3_ir_f105w_coars	0.12”/pixel	hst_skycell-p1889x07y19_wfc3_ir_f105w_coarse-all_drz.fits
wfc3_ir_f105w	0.04”/pixel	hst_skycell-p1889x07y19_wfc3_ir_f105w_all_drz.fits
wfc3_ir_f125w_coars	0.12”/pixel	hst_skycell-p1889x07y19_wfc3_ir_f125w_coarse-all_drz.fits
wfc3_ir_f125w	0.04”/pixel	hst_skycell-p1889x07y19_wfc3_ir_f125w_all_drz.fits
wfc3_ir_f160w_coars	0.12”/pixel	hst_skycell-p1889x07y19_wfc3_ir_f160w_coarse-all_drz.fits
wfc3_ir_f160w	0.04”/pixel	hst_skycell-p1889x07y19_wfc3_ir_f160w_all_drz.fits
acs_wfc_f850lp	0.04”/pixel	hst_skycell-p1889x07y19_acs_wfc_f850lp_all_drc.fits
acs_wfc_f775w	0.04”/pixel	hst_skycell-p1889x07y19_acs_wfc_f775w_all_drc.fits

A full SkyCell would cover an area on the sky of approximately 0.2deg x 0.2deg with a WCS defined as:

```
Number of WCS axes: 2
CTYPE : 'RA---TAN'  'DEC--TAN'
CRVAL : 180.0  26.0
CRPIX : 96492.0  -160812.0
CD1_1 CD1_2  : -1.1111111111111112e-05  0.0
CD2_1 CD2_2  : 0.0  1.1111111111111112e-05
NAXIS : 21954  21954
```

SkyCell Subarray Specification

SkyCell layers would normally result in arrays that take up 1.8Gb each, or nearly 4Gb for the entire FITS file. In addition, most of a typical SkyCell layer will be empty due to the small size of most detectors, especially the WFC3/IR, ACS/HRC and ACS/SBC detectors which are only 1024x1024 arrays. As a result, each SkyCell is evaluated to define only a common subarray size that covers ALL HST data for the SkyCell regardless of the layer, then uses that to define the smallest WCS specification possible for the SkyCell. The data from **p1889x07y19** actually only covers about 1/4 of the entire SkyCell resulting in a WCS defined as:

```
>>> wcs1889 = HSTWCS('hst_skycell-p1889x07y19_acs_wfc_f775w_all_drc.fits', ext=1)
>>> print(wcs1889)
```

(continues on next page)

(continued from previous page)

```
WCS Keywords
Number of WCS axes: 2
CTYPE : 'RA---TAN'  'DEC--TAN'
CRVAL : 180.0  26.0
CRPIX : 94360.0  -171093.0
CD1_1 CD1_2  : -1.1111111111111e-05  0.0
CD2_1 CD2_2  : 0.0  1.11111111111111e-05
NAXIS : 10840  11672
```

We can see how the exposures land in the SkyCell as defined with this subarray WCS.

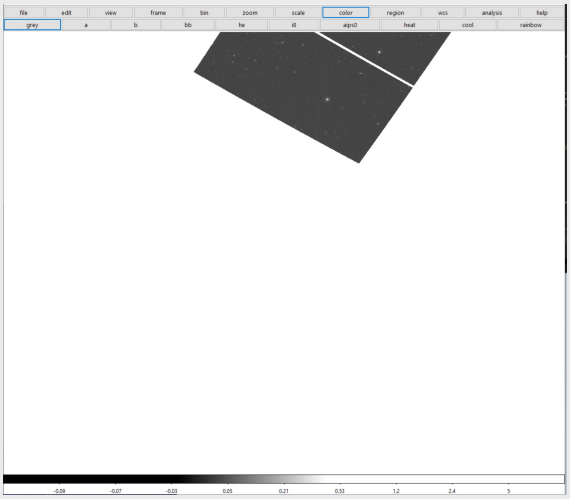


Fig. 5: All the WFC3/UVIS F775W exposures that overlap SkyCell **p1889x07y19**.

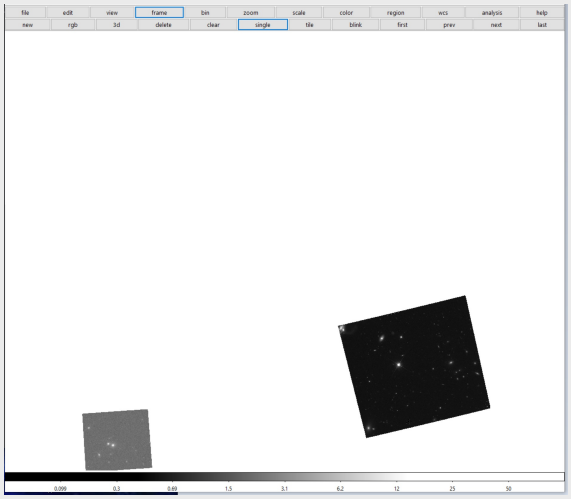


Fig. 6: All the WFC3/IR F105W exposures that overlap SkyCell **p1889x07y19**.

These figures demonstrate how the SkyCell subarray has been defined to cover only the exposures that overlap this SkyCell while minimizing the amount of empty space in these layers making these products as small as possible without resorting to compression.

Note: These layers are defined as subarrays of the entire SkyCell WCS, which are just subarrays of the ProjectionCell. As a result, any position in these layers refers to the exact same position on the sky as defined in the SkyCell or ProjectionCell. A source at (x,y)=(7936, 11133) will have the same sky coordinates regardless of what SkyCell layer it was measured in, including an array defined for the entire SkyCell.

File Format

The SkyCell layer mosaics get generated using AstroDrizzle using a custom set of rules for defining what keywords go into the primary and science extension headers. These files have the same set of extensions as all other drizzled products; namely, PRIMARY, SCI, WHT, CTX and HDRTAB.

MVM-specific keywords

MVM mosaics, by definition, include the contributions of many exposures (in most cases) potentially taken at many different times for a specific section of the sky. Header keywords have been defined to provide some information on the unique characteristics of these mosaics and the contribution of the exposures to the mosaic. This unique set of keywords defined in the PRIMARY headers of MVM mosaics includes:

Keyword	Description
SCELLID	ID of the SkyCell this mosaic covers
NPIXFRAC	Fraction of pixels across the full SkyCell which has been observed by HST
MEAN-EXPT	Mean exposure time of pixels which have been observed by HST
MEDEXPT	Median exposure time of pixels which have been observed by HST
MEAN-NEXP	Mean number of HST exposures for the pixels which have been observed by HST
MEDNEXP	Median number of HST exposure for the pixels which have been observed by HST
MEAN-WHT	Mean weight (typically exposure time) of pixels observed by HST
MEDWHT	Median of weights (typically exposure time) of pixels observed by HST

In addition, some keywords typically found in standard pipeline product headers or the headers of SVM mosaics have been removed from the headers of MVM products. These deleted keywords can be found in the HDRTAB extension for each of the input exposures, but make no sense for MVM products. One example would be the 'IPPPSSOO' keyword which gives the 'ipppssoo' value for a single exposure or association product the input exposures had, yet MVM mosaics consist of multiple inputs with many different values of 'ipppssoo'. The full list of keywords which were removed can be found in the HISTORY keywords of the MVM product PRIMARY header where the rules file used for defining the MVM headers gets reported.

Artifacts

There are a number of issues that can arise when generating SkyCell mosaics. Every effort is made during pipeline processing to minimize or avoid these issues where possible, but some mosaics are unavoidably affected by these issues.

Mis-alignment

One of the primary benefits of SkyCell mosaic image is learning how observations taken at different times and using different filters relate to each other. The mosaics all share the same pixel definitions which allow for direct comparisons of the data across all the layers of a SkyCell. However, the placement of the exposures in the SkyCell depends on how the WCS was defined for each exposure. Unfortunately, due to the objects in the field of view for an exposure or the size of the field of view of the exposure, it may not be possible to align the exposure to the same astrometric catalog as the rest of the exposures in the SkyCell. This can lead to mis-alignment between the exposures on the order of a few pixels. If this mis-aligned exposure overlaps another exposure aligned to GAIA in a SkyCell mosaic, then it can result in visible blurring or double-images in the final mosaic.

This can be seen when examining the SkyCell mosaic WFC3/UVIS F555W layer for SkyCell **p0498x16y19**. The WCS has been defined based on different astrometric catalogs for one of the visits of this source. This results in 'blurred' sources being seen in the overlap between exposures from different visits.

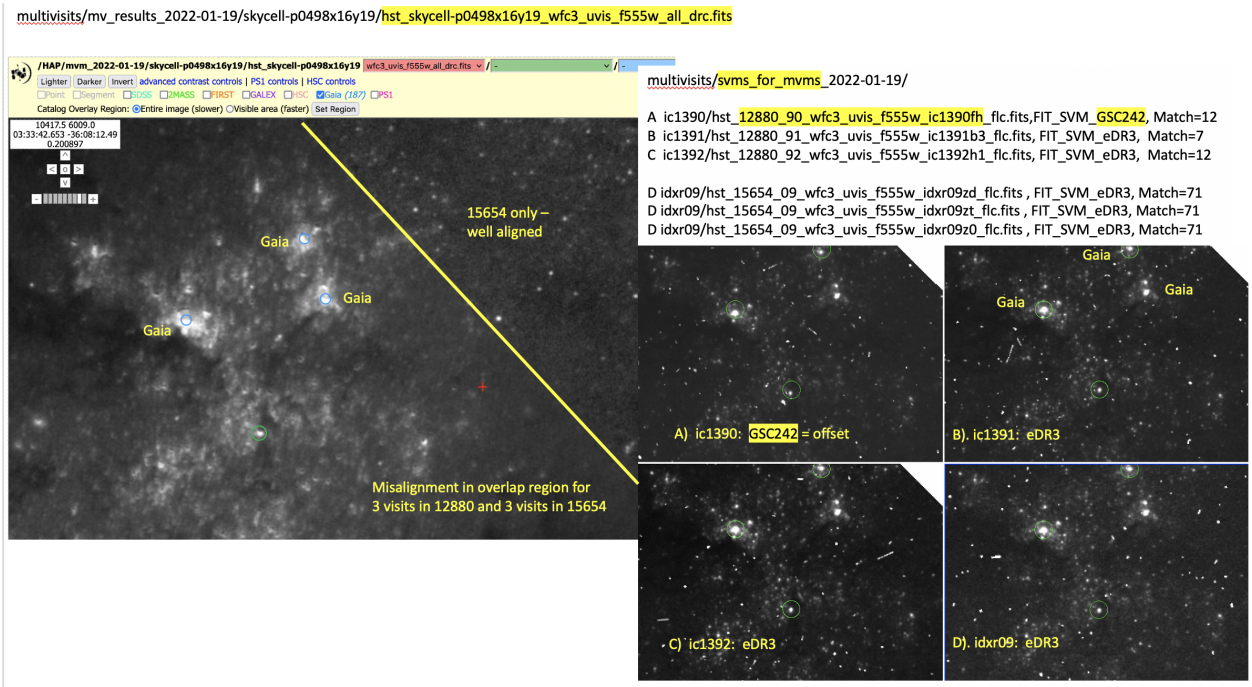


Fig. 7: All the WFC3/UVIS F555W exposures that overlap SkyCell **p0498x16y19** illustrating how mis-alignment between visits can result in 'blurred' sources in region of overlap. [Image courtesy of J. Mack (STScI/ACS Instrument team)].

Loss of Lock

Another issue that can show up in SkyCell mosaics results from HST slewing across the sky while the exposure was being taken. This can happen when HST loses lock on the guide stars used to point the telescope or intentionally when the proposer requested exposures be taken in 'SCAN' mode. No reliable method currently exists to identify such exposures prior to creating mosaics with them, resulting in exposures where the sources are trailed across the image.

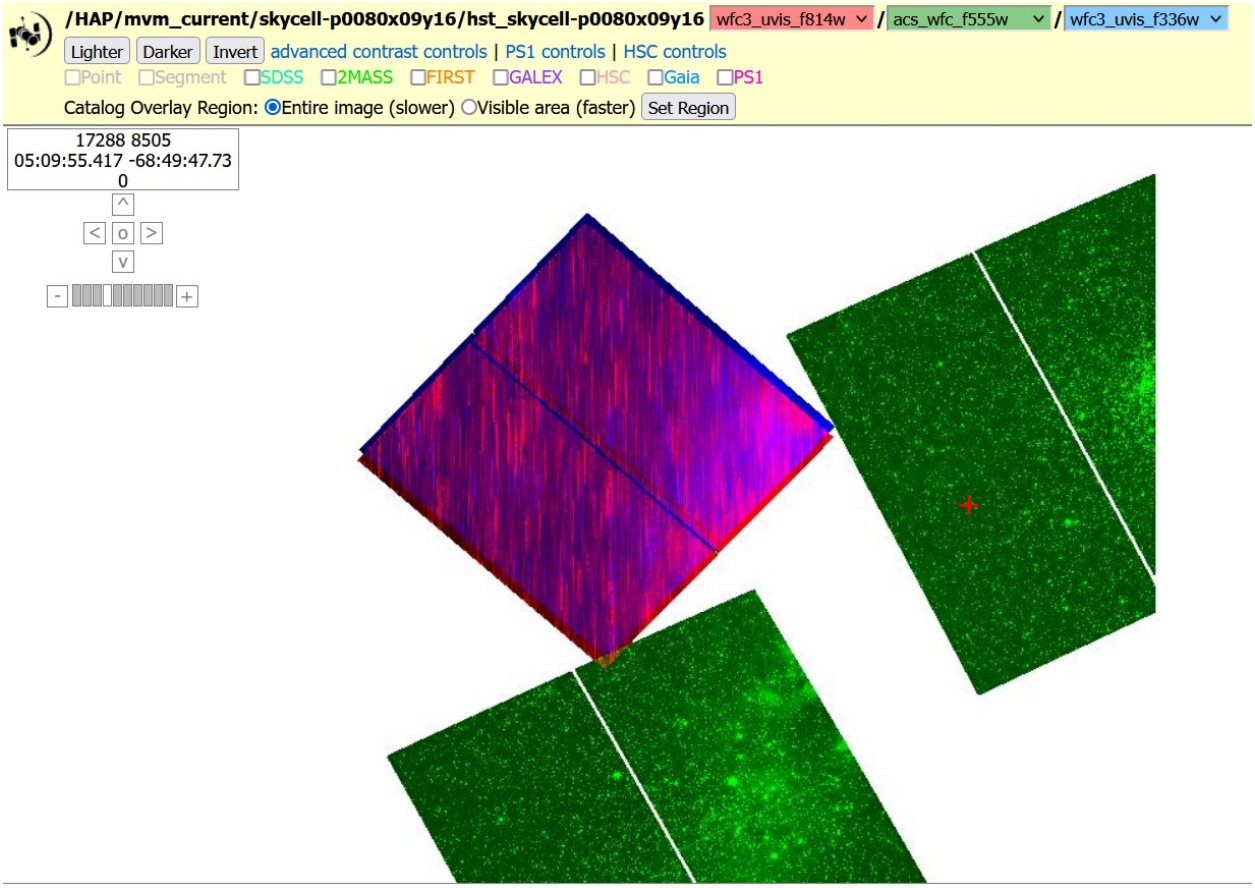


Fig. 8: SkyCell **p0080x09y16** mosaic showing the effects of loss of lock or SCAN mode data being included in the output image.

Fortunately, these observations stand out as very dramatic regions of the MVM mosaic where no recognizable sources can be found, only a series of streaks as seen in the example. There are some exposures where SCAN mode was used to observe extremely bright sources, however, many of the examples in the archive are the result of unexpected tracking problems by HST.

The best option for anyone wanting to explore the region affected by such observations would be to get the list of exposures that contribute to the MVM mosaic and reprocess the SkyCell interactively without including the problematic exposures. Eventually, these observations may be removed from the archived products, but until that time comes, manual reprocessing would be the suggested means for studying SkyCells affected by such exposures.

Alignment Accuracy Across the SkyCell

Every effort gets made to align all exposures to a GAIA-based coordinate system. Some exposures, though, just can not be aligned (or aligned in an automated fashion) to the GAIA system due to any number of reasons. In addition, not all exposures can be aligned to the same GAIA-based catalog of sources as the exposure may not include enough GAIA sources for alignment, but may contain sources measured by other projects (like Pan-STARRS) that have been subsequently fit to the GAIA coordinate system. This will result in a larger uncertainty for the coordinates for those sources. The best available WCS for any given exposure eventually gets defined by the SVM processing performed by the HST calibration pipeline and these aligned products then get used as inputs to generate the MVM mosaics.

MVM mosaics include as many exposures as possible but due to these reasons, a given MVM mosaic can contain exposures fit to different astrometric catalogs. Exposures which do not overlap other exposures in the MVM mosaic can only use the WCS defined during SVM processing and that may not be the most accurate GAIA-based catalog available. This can result in errors in the relative alignment between those exposures and the rest of the exposures in the mosaic which have been aligned to the most accurate catalog available. These errors may not be large (less than a few pixels in nearly all cases), but care must be taken to interpret the positions of sources across a SkyCell due to such effects.

Effects of Proper Motion

HST has been taking images for over 30 years, with ACS being active since 2002 and WFC3 having been installed in 2009. As a result, there have been some fields which have been observed multiple times over those decades allowing HST with it's high resolution to observe the stellar proper motions. Any source with proper motions of a few milli-arcseconds motion per year or more could potentially be observed moving when comparing images taken over the life of HST. Such sources can be found in observations of sources like nearby globular clusters (e.g., 47Tuc or Omega Cen) or open clusters (like M35).

Unfortunately, this makes aligning images of such high proper motion sources difficult to interpret. Measurements of proper motions for each source have been included in the catalog information for sources in the GAIA-based astrometric catalogs for all catalogs derived using GAIADR2 or later. The HST images taken during the same visit are then aligned to the GAIA-based catalog, using the GAIAeDR3 catalog by default through 2022 during the SVM processing. This SVM alignment generates a consistent fit for the proper motions at a single epoch. These SVM-aligned exposures are then used, without further alignment, to generate the final MVM mosaic of all exposures taken over the life of HST. As a result, each epoch represents a single snapshot of the sources at one time. The sources in the field with large proper motions, on the other hand, are not going to align as they have moved from one epoch. This will result in those sources showing up with smeared PSFs or even double-exposures in the final MVM mosaic image while the background sources (typically background galaxies or globular clusters) will be well aligned.

These errors can actually be a good way to spot high proper motion sources for later study. In addition, improvements to the MVM processing code are being developed to allow for such fields to be processed such that observations taken at different times end up in their own MVM mosaic rather than having all the exposures combined by default. This would enable more direct measurement of such high proper motion sources, eventually. The initial set of MVM products available through the archive, though, will be comprised of layers consisting of all exposures from all times for each detector/filter combination as indicated by the 'all' in the final MVM product filename.

Utilities for Manipulating MVM Products

The *hapcut_utils* module contains utilities which are intended to be used with Multi-visit Mosaic (MVM) data and to be imported into other Python programs or a Python session for interactive data analysis. The utilities are wrappers around lower-level functionality which actually perform the core tasks. The core tasks referenced here are astroquery, used for MAST inquiries and retrievals, and astrocut, used to create the cutouts and cutout combinations. The *hapcut_utils* role is to customize the use of these core tasks specifically for working with MVM products which can be large and unwieldy. At this time there are four utilities which are designed to be invoked in sequence as is evident from their function signatures, return values, and output products: `mvm_id_filenames`, `mvm_retrieve_files`, `make_the_cut`, and `mvm_combine`.

A brief overview of the utilities is described in this list.

1. `mvm_id_filenames`: Query MAST to identify all MVM files which overlap the specified cutout region on the sky
2. `mvm_retrieve_files`: Download all requested MVM files from MAST to the working directory
3. `make_the_cut`: Extract cutouts from all specified MVM files
4. `mvm_combine`: Create a mosaic from the specified cutouts that span multiple SkyCells and fit the qualifications

MVM Cutout Utilities

mvm_id_filenames

This utility is invoked by providing sky coordinates and a cutout box size. The sky coordinates represent the central location about which the search for data of interest is to be done. The cutout box size represents the search box *extent* encompassing the sky coordinates. The utility retrieves from MAST a table containing MVM filenames for ACS and WFC3 detectors, as well as other relevant image specific information, one row per filename. While the utility returns an astropy table object, an ascii ECSV file is also written to disk with the output filename being constructed from the query coordinates and box size parameters. The output filename is of the form:

```
mvm_query-ra<###>d<####>-dec<n|s><##>d<####>_<radius>_cutout.ecsv
```

The `ra<###>d<####>` and `dec<n|s><##>d<####>` are the right ascension and declination, respectively, in decimal degrees with `<n|s>` representing north or south of the celestial equator. The `<radius>` is the computed diagonal of the cutout box which is basically the square root of the sum of the box sides squared divided by two.

The in-memory table or the ascii file can be refined by the user to suit specific needs. The MVM filenames within the query file are self-descriptive. The filter-level drizzled filename is of the form:

```
hst_skycell-p<PPPP>x<XX>y<YY>_<instr>_<detector>_<filter>_<label>_dr[cz].fits
```

where `-p` is followed by the four digit projection cell value, and `x` and `y` are each followed by a two digit sky cell value. The possible `label` values at this time are `all` or `coarse-all` where `all` indicates all

exposures while `coarse-all` indicates all the exposures at a coarse platescale (i.e., the WFC3/IR platescale of 0.12"/pixel) in the visit were used in the generation of the product.

While the standard pipeline processing does not generate MVM exposure-level, drizzled products, the user can activate this capability when performing custom processing. As such, the MAST query will not contain any exposure-level drizzled product filenames. However, an example of an exposure-level drizzled filename created by custom processing would be of the form:

```
hst_skycell-p<PPPP>x<XX>y<YY>_<instr>_<detector>_<filter>_<label>-ipppssoo_
↪dr[cz].fits
```

An example of a real MVM filter-level, drizzled filename is:

```
hst_skycell-p0081x14y14_wfc3_uvis_f225w_all_drc.fits
```

mvm_retrieve_files

This utility is designed to use the in-memory table generated by `mvm_id_filenames` and retrieve any file listed in the table from MAST. Since the images can be quite large, the `clobber` parameter is set to False by default, indicating no download is done for a filename that already exists on disk. The utility returns a Python list of filenames on disk available for further analysis.

make_the_cut

The `make_the_cut` utility will perform the FITS cutouts on the input MVM filter- and exposure-level images. This MVM customized utility is layered on `fits_cut`, a function which is part of Astrocut, a Python package for making astronomical cutouts (https://astrocut.readthedocs.io/en/latest/). The `make_the_cut` function is invoked by providing the list of files on disk for which one wants to make cutouts, as well as the sky coordinates, and the cutout box size. Ideally, the sky coordinates and cutout box size are the same values used for the `mvm_id_filenames` utility. For each input file designated for a cutout, there will be a corresponding output file. Since both the SCI and WHT extensions of the input files are actually cut, individual output cutout FITS files will contain two image extensions, a SCI followed by a WHT. The cutouts are explicitly generated by looping over the input file list in case one of the files experiences an exception so the remaining files in the list still have the opportunity to create a cutout. Each filter-level output filename will be of the form:

```
hst_cutout_skycell-p<pppp>-ra<##>d<####>-dec<n|s><##>d<####>_instrument_detector_
↪filter[_platescale].fits
```

Each exposure-level filename generated by custom user processing will be of the form:

```
hst_cutout_skycell-p<pppp>-ra<##>d<####>-dec<n|s><##>d<####>_instrument_detector_
↪filter[_platescale]-ipppssoo.fits
```

The components of the output filename follow the pattern discussed for `mvm_id_filenames`. The additional `platescale` component has the possible value of `coarse`. When there is no `platescale` component present in the filename, the `platescale` has the implicit default value of `fine`. The `fine` platescale is

0.04"/pixel and is the standardized platescale for all MVM products. The platescale of `coarse` is only applicable to the WFC3/IR data.

The primary header data unit (PHDU) from the input file is retained as the PHDU for the output cutout file with an updated FILENAME keyword and a few additional keywords. The new keywords RA_OBJ/DEC_OBJ were added by the Astrocut utility representing the right ascension and declination of the object in decimal degrees. Another new keyword is ORIG_FLE which represents the original input filename from which the cutout was extracted. In each output file, the image extensions have been explicitly named for for clarity.

```
>>> from astropy.io import fits
>>> hdu= fits.open("hst_cutout_skycell-p0081x14y15-ra84d8208-decs69d8516_wfc3_
↪uvis_f275w.fits")
>>> hdu.info()
Filename: hst_cutout_skycell-p0081x14y15-ra84d8208-decs69d8516_wfc3_uvis_f275w.
↪fits
No.    Name      Ver    Type      Cards   Dimensions   Format
  0  PRIMARY       1 PrimaryHDU     872   ()
  1  SCI           1 ImageHDU       106   (12500, 12500)   float32
  2  WHT           1 ImageHDU        56   (12500, 12500)   float32
```

mvm_combine

The final MVM utility at this time is `mvm_combine` and is designed to combine filter-level cutout images originating from multiple skycells where all the images were acquired with the same detector and using the same filter. The utility will also combine exposure-level cutout images from multiple skycells where the images were acquired with the same detector, using the same filter, and sharing the same ipppssoo and created by user custom processing. Put more straightforwardly, the cutout images must have originated from the same MVM drizzled image which spanned the border between multiple skycells. Cutout images in the input list which do not adhere to these noted requirements will trigger a message of explanation and be ignored for the combination process.

The `mvm_combine` utility is invoked by providing a list of cutout files, and the utility will handle segregating the files into filter- and exposure-level sets and determining whether or not the cutout file is eligible for the combination process. Once all the input images have been prepared, the mvm_combine will call the Astrocut function `CutoutsCombiner` to handle the actual combination task. At this time, only the default *mean* combiner is available for use where every output pixel is the mean of all input pixels that have data at that point.

Since both the SCI and WHT extensions of the input files are combined, individual output combined FITS files will contain two image extensions, a SCI followed by a WHT. The utility writes to the specified output directory one combined file per set of compatible images. The output filenames are constructed from the RA, Dec, instrument, detector, filter, and exposure as appropriate. Each filter-level output filename will be of the form:

```
hst_combined_skycell-ra<##>d<####>-dec<n|s><##>d<####>_instrument_detector_
↪filter.fits
```

Each exposure-level filename where the cutout images were generated by custom user processing will be of the form:

```
hst_combined_skycell-ra<##>d<####>-dec<n|s><##>d<####>_instrument_detector_
↪filter_ipppssoo.fits
```

The keyword, FILENAME, has been added to the PHDU to self-document the name of the file. The Extension Header Data Units (EHDUs) contain custom keywords which document the filenames and extension names of the originating files. These custom keywords are created by the `CutoutsCombiner` and are of the form `Fnn_[K|V|C]mm` representing the keywords EXTNAMEs and ORIG_FLEs from the input cutout files. The `nn` represents the header number, and `mm` represents a count of the custom keywords. For example,

```
>>> hdu = fits.open("hst_combined_skycells-ra84d9402-decs69d8514_acs_wfc_f658n.
↪fits")
>>> hdu.info()
Filename: hst_combined_skycells-ra84d9402-decs69d8514_acs_wfc_f658n.fits
No.    Name      Ver    Type      Cards   Dimensions   Format
  0  PRIMARY       1 PrimaryHDU      10   ()
  1  SCI           1 ImageHDU       167   (250, 250)   float64
  2  WHT           1 ImageHDU        70   (250, 250)   float64

where the first extention or combined SCI extension header contains
O_EXT_NM= 'SCI     '           / Original extension name.
ORIG_EXT=                    1 / Extension in original file.
F01_K01 = 'EXTNAME '           / Keyword
F01_V01 = 'SCI'                / Value
F01_C01 = 'extension name'     / Comment
F02_K01 = 'EXTNAME '           / Keyword
F02_V01 = 'SCI'                / Value
F02_C01 = 'extension name'     / Comment
F01_V02 = 'hst_skycell-p0081x14y15_acs_wfc_f658n_all_drz.fits' / Value
F01_C02 = 'Original image'     / Comment
F02_K02 = 'ORIG_FLE'           / Keyword
F02_V02 = 'hst_skycell-p0081x14y14_acs_wfc_f658n_all_drc.fits' / Value
F02_C02 = 'Original image'     / Comment
EXTNAME = 'SCI'
```

and similarly for the second or combined WHT extension.

A Full Worked Example

In the example below, the ellipses ("…") are used to indicate the omission of output similar to what has already been documented for the reader. Note that as of this writing, there are actually no MVM files in MAST upon which to perform a query or request a retrieval so details on output messages may vary from the true output.

```
>>> from drizzlepac.haputils import hapcut_utils as hu
>>> from astropy.coordinates import SkyCoord
>>> query_table = hu.mvm_id_filename(SkyCoord(84.9208, -69.1467, unit="deg"),␣
↪[256, 256])
2021295152553 INFO src=hapcut- Radius for query: 182.0 arcsec.
2021295152553 INFO src=hapcut- Performing query for ACS images.
2021295152621 INFO src=hapcut- Performing query for WFC3 images.
2021295152646 INFO src=hapcut- Get the product list for all entries in the query␣
↪table.
2021295152646 INFO src=hapcut- Filter the product list table for only ['DRZ',
↪'DRC'] filenames.
2021295152646 INFO src=hapcut- Writing out the MVM product list table to mvm_
↪query-ra84d9207-decs69d1467_182_cutout.ecsv.
2021295152646 INFO src=hapcut- Number of entries in table: 26.
```

The `mvm_retrieve_files` simply gets the requested files, in this case every file in the query_table, from MAST unless the files are already found to be resident on disk. It should be noted any individual MVM file could be rather large and unwieldy.

```
>>> requested_files = hu.mvm_retrieve_files(query_table)
```

The `make_the_cut` function loops over the input list of files and tries to make the requested cutout. If a cutout cannot be made, an error message will be generated, and the function will continue processing. All cutout files have the prefix `hst_cutout`.

```
>>> cutout_files = hu.make_the_cut(requested_files, SkyCoord(84.9208, -69.1467,␣
↪unit="deg"), [256, 256])
Number of input files: 1
Cutting out [1, 2] extension(s)
Center coordinate: 84.9208 -69.1467 deg
Cutout size: [256. 256.] arcsec

Cutting out hst_skycell-p0081x14y14_acs_wfc_f658n_all_drc.fits
Original image shape: (21953, 11503)
xmin,xmax: [ 8093 14493]
ymin,ymax: [18554 24954]
Image cutout shape: (6400, 6400)
Original image shape: (21953, 11503)
xmin,xmax: [ 8093 14493]
ymin,ymax: [18554 24954]
```

(continues on next page)

(continued from previous page)

```
Image cutout shape: (6400, 6400)
Returning cutouts as individual FITS
Total time: 4.3 sec

...

2021295103421 INFO src=hapcut- Cutout FITS filename: ./hst_cutout_skycell-
↪p0081x14y14-ra84d9207-decs69d8533_wfc3_ir_f110w_coarse-iby03fg6.fits
2021295103422 INFO src=hapcut- Cutout FITS filename: ./hst_cutout_skycell-
↪p0081x14y14-ra84d9207-decs69d8533_acs_wfc_f658n.fits
```

The `mvm_combine` function loops over the input list of cutout files and organizes the data according to instrument/detector and filter for filter-level combinations. The data is organized by instrument/detector, filter, and ipppssoo for exposure-level combinations. Images in the input list which do not share an instrument/detector and filter with any other image will be ignored. Individual exposures from a single skycell will also be ignored.

```
>>> hu.mvm_combine(cutout_files)
2021295104231 INFO src=hapcut- Input cutout files:
2021295104231 INFO src=hapcut- File: hst_cutout_skycell-p0081x14y14-ra84d9207-
↪decs69d8533_acs_wfc_f555w-jby001r0.fits
2021295104231 INFO src=hapcut- File: hst_cutout_skycell-p0081x14y14-ra84d9207-
↪decs69d8533_acs_wfc_f555w.fits
2021295104231 INFO src=hapcut- File: hst_cutout_skycell-p0081x14y15-ra84d9207-
↪decs69d8533_acs_wfc_f814w-jbp505jg.fits
2021295104231 INFO src=hapcut- File: hst_cutout_skycell-p0081x14y15-ra84d9207-
↪decs69d8533_acs_wfc_f814w.fits

...

2021295104231 INFO src=hapcut- === Combining filter-level files ===
2021295104231 WARNING src=hapcut- There is only one file for this detector/
↪filter[/ipppssoo] combination, so there is nothing to combine.
2021295104231 WARNING src=hapcut- File ['hst_cutout_skycell-p0081x14y14-
↪ra84d9207-decs69d8533_acs_wfc_f555w.fits'] will be ignored for combination␣
↪purposes.

2021295104231 WARNING src=hapcut- There is only one file for this detector/
↪filter[/ipppssoo] combination, so there is nothing to combine.
2021295104231 WARNING src=hapcut- File ['hst_cutout_skycell-p0081x14y15-
↪ra84d9207-decs69d8533_acs_wfc_f814w.fits'] will be ignored for combination␣
↪purposes.

2021295104231 INFO src=hapcut- Combining the SCI and then the WHT extensions of␣
↪the input cutout files.
2021295104237 INFO src=hapcut- The combined output filename is ./hst_combined_
```

(continues on next page)

(continued from previous page)

```
↪skycells-ra84d9207-decs69d8533_acs_wfc_f658n.fits.

...

2021295104242 INFO src=hapcut-
2021295104242 INFO src=hapcut- === Combining exposure-level files ===
2021295104242 WARNING src=hapcut- There is only one file for this detector/
↪filter[/ipppssoo] combination, so there is nothing to combine.
2021295104242 WARNING src=hapcut- File ['hst_cutout_skycell-p0081x14y14-
↪ra84d9207-decs69d8533_acs_wfc_f555w-jby001r0.fits'] will be ignored for␣
↪combination purposes.

...

2021295104242 INFO src=hapcut- Combining the SCI and then the WHT extensions of␣
↪the input cutout files.
2021295104246 INFO src=hapcut- The combined output filename is ./hst_combined_
↪skycells-ra84d9207-decs69d8533_wfc3_ir_f110w_iby03fg4.fits.

...

2021295104255 INFO src=hapcut- Cutout combination is done.
```

3.2.4 Hubble Advanced Products API

Drizzlepac has evolved over time from not only being responsible for creating the calibration pipeline drizzled products (i.e., ipppssoot_dr[c|z].fits), but also to being responsible for the generation of the two types of Hubble Advanced Products (HAP). The HAP products are Single Visit Mosaics (SVMs) and Multiple Visit Mosaics (MVMs) and the creation of these products has been incorporated into the calibration pipeline processing. SVM and MVM drizzled products are distingushed from pipeline drizzled counterparts and each other through their filenames, as well as internal contents.

SVM: hst_<propid>_<obsetid>_<instr>_<detector>_<filter>_<ipppssoo>_dr[c|z].fits) MVM: hst_skycell_p<PPPP>x<XX>y<YY>_<instr>_<detector>_<filter>_<label>_dr[c|z].fits)

where PPPP = 4-digit projection cell number XX, YY = 2-digit sky cell numbers

runastrodriz.py is the module to control operation of AstroDrizzle to remove distortion and combine HST images.

runsinglehap.py is the module which controls the SVM processing.

runmultihap.py is the module which controls the MVM processing.

Sucessful code will result in an exit code of 0. If there is an error, the exit code will be non-zero. An exit code of 55 is for data containing only SBC or HRC data, and an exit code of 65 is for "no viable data".

Fig. 9: This cutout is hst_cutout_skycell-p0081x14y14-ra84d9207-decs69d8533_acs_wfc_f658n.fits scaled in size by 75%. The region in common with skycell-p0081x14y15 is in the upper-right corner.

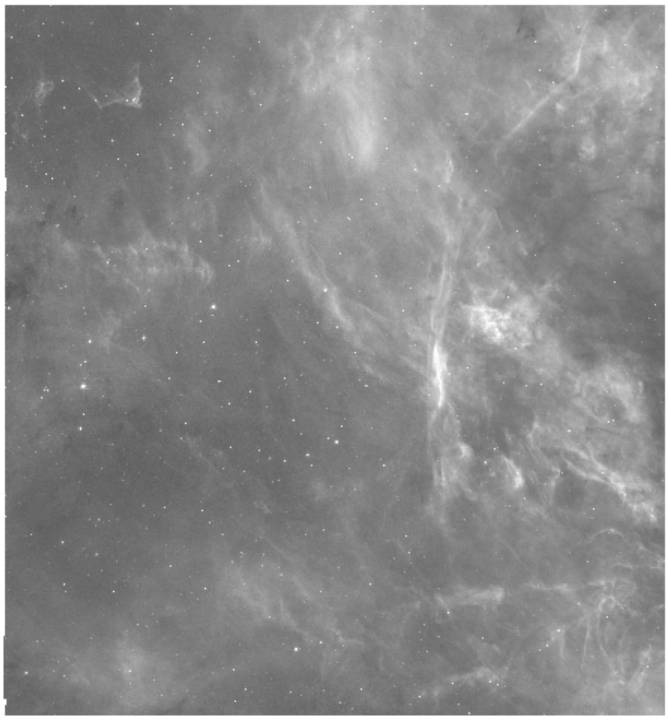


Fig. 10: This cutout is hst_cutout_skycell-p0081x14y15-ra84d9207-decs69d8533_acs_wfc_f658n.fits scaled in size by 75%. The region in common with skycell-p0081x14y14 is in the lower-right corner.

HAP API

These modules provide the basic functionality used to process automatically data using this package to apply the distortion models to the WCS of HST observations and to verify the alignment of the observations.

align

This script is a modernized implementation of tweakreg.

drizzlepac.align.**perform_align**(*input_list*, *catalog_list*, *num_sources*, *archive=False*, *clobber=False*, *debug=False*, *update_hdr_wcs=False*, *result=None*, *runfile='temp_align.log'*, *print_fit_parameters=True*, *print_git_info=False*, *output=False*, *headerlet_filenames=None*, *fit_label=None*, *product_type=None*, ***alignment_pars*)

Actual Main calling function.

This function performs a posteriori astrometric fits to the images specified in the input_list. The images are fit to all the catalogs listed in the catalog_list parameter with the results being saved and returned as an Astropy Table object. This allows the user to select the solution that is most appropriate.

Parameters

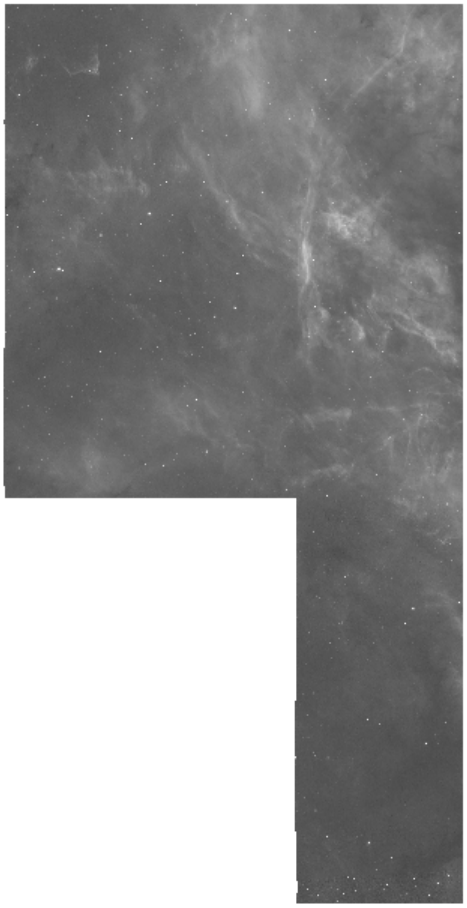


Fig. 11: This figure is the combination of the two cutouts scaled in size by 75%.

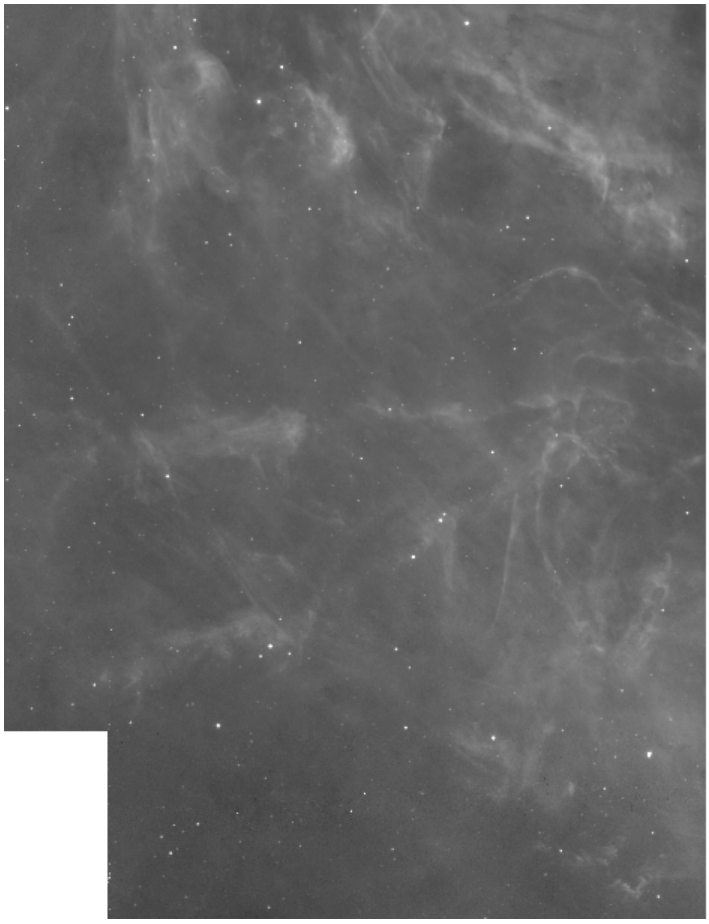


Fig. 12: Finally, this figure is the combination of the two cutouts scaled in size by 75% and zoomed in to the overlap region.

input_list
[list] List of one or more IPPSSOOTs (rootnames) to align.

catalog_list
[list] Set of astrometric catalogs which should be used as references for fitting the input images. A separate fit will be performed for each catalog specified. The catalog name will also be used as part of the output `WCSNAME` value for the fit determined from that catalog.

num_sources
[int, None] Maximum number of **brightest sources per chip** which will be used for cross-matching and fitting. If set to None, all sources will be used.

archive
[Boolean] Retain copies of the downloaded files in the astroquery created sub-directories?

clobber
[Boolean] Download and overwrite existing local copies of input files?

debug
[Boolean] Attempt to use saved sourcelists stored in pickle files if they exist, or if they do not exist, save sourcelists in pickle files for reuse so that step 4 can be skipped for faster subsequent debug/development runs??

update_hdr_wcs
[Boolean] Write newly computed WCS information to image image headers?

result: Table
name of variable to be updated by subroutine run.

runfile
[string] log file name

print_fit_parameters
[Boolean] Specify whether or not to print out FIT results for each chip.

print_git_info
[Boolean] Display git repository information?

output
[Boolean] Should utils.astrometric_utils.create_astrometric_catalog() generate file 'ref_cat.ecsv' and should the alignment source catalogs get written out to files?

headerlet_filenames
[dictionary, optional] dictionary that maps the flt/flc.fits file name to the corresponding custom headerlet filename.

fit_label
[str] String to use as a unique tag for the WCS solutions generated by these fits. This tag will be inserted between the `-FIT` and the catalog name in the `WCSNAME` keyword value; for example, ``fit_label="User"`` will result in fits to GAIAedr3 with names ending in `-FIT_User_GAIAedr3`.

alignment_pars
[dictionary or keyword args] keyword-arg parameters containing user-specified values for the parameters used in source identification and alignment which should replace the default values found in the JSON parameter files in `drizzlepac.pars.hap_pars` based on the instrument and detector. The code will look for default values for all the parameters in the JSON parameter files using `get_default_pars`. For example, should the user feel it would be more successful to only look out to a radius of 25 pixels during alignment, the user could simply specify `searchrad=25`.

Returns

filtered_table
[Astropy Table] Table gets updated to contain processing information and alignment results for every raw image evaluated

drizzlepac.align.**determine_fit_quality**(*imglist*, *filtered_table*, *catalogs_remaining*, *align_pars*, *print_fit_parameters=True*, *loglevel=0*, *runfile='temp_align.log'*)

Determine the quality of the fit to the data

Parameters

imglist
[list] output of interpret_fits. Contains sourcelist tables, newly computed WCS info, etc. for every chip of every valid input image. This list should have been updated, in-place, with the new RMS values; specifically,

- 'FIT_RMS': RMS of the separations between fitted image positions and reference positions
- 'TOTAL_RMS': mean of the FIT_RMS values for all observations
- 'NUM_FITS': number of images/group_id's with successful fits included in the TOTAL_RMS

These entries are added to the 'fit_info' dictionary.

filtered_table
[object] Astropy Table object containing data pertaining to the associated dataset, including the doProcess bool. It is intended this table is updated by subsequent functions for bookkeeping purposes.

catalogs_remaining
[bool] Specify whether additional catalogs remain to be fit against.

align_pars
[dict] Parameters from the appropriate '_align_all.json' parameter file

print_fit_parameters
[bool] Specify whether or not to print out FIT results for each chip

log_level
[int, optional] The desired level of verboseness in the log statements displayed on

the screen and written to the .log file. Default value is 20, or 'info'.

runfile
[string] log file name

Returns

max_rms_val
[float] The best Total rms determined from all of the images

num_xmatches: int
The number of stars used in matching the data

fit_quality
[int]

fit quality category:

- 1 = valid solution with rms < 10 mas
- 2 = Valid but compromised solution with rms < 10 mas
- 3 = Valid solution with RMS >= 10 mas
- 4 = Valid but compromised solution with RMS >= 10 mas
- 5 = Not valid solution

filtered_table
[object] modified filtered_table object

fit_status_dict
[dictionary]

Dictionary containing the following:

- overall fit validity (Boolean)
- total (visit-level) RMS value in mas (float)
- number of matched sources (int)
- fit compromised status (Boolean)
- reason fit is considered 'compromised' (only populated if "compromised" field is "True")

haputils.astrometric_utils

Utilities to support creation of astrometrically accurate reference catalogs

The function, create_astrometric_catalog, allows the user to query an astrometric catalog online to generate a catalog of astrometric sources that should fall within the field-of-view of all the input images.

This module relies on the definition of an environment variable to specify the URL of the astrometric catalog to use for generating this reference catalog.

```
ASTROMETRIC_CATALOG_URL  -- URL of web service that can be queried to
                            obtain listing of astrometric sources,
                            sky coordinates, and magnitudes.
```

drizzlepac.haputils.astrometric_utils.**create_astrometric_catalog**(*inputs*, *catalog='GAIAedr3'*, *output='ref_cat.ecsv'*, *gaia_only=False*, *table_format='ascii.ecsv'*, *existing_wcs=None*, *num_sources=None*, *use_footprint=False*, *full_catalog=False*, *user_epoch='match'*, *log_level=20*)

Create an astrometric catalog that covers the inputs' field-of-view.

This function will return a table containing sources derived from the selected astrometric catalog specified by the user for the field-of-view covered by the list of input images.

Note: The output table generated by this function will have a pre-defined set of columns with common names for catalogs included in SUPPORTED_CATALOGS. Columns which don't exist in the catalog will be added with null (None) values. The common column names are:

- **RA, DEC** : RA, DEC of source
- **RA_error, DEC_error** : error in RA, DEC position
- **objID** : object ID from catalog
- **pmra, pmdec** : Proper motion in RA and DEC
- **pmra_error, pmdec_error** : errors in the proper motion
- **mag** : reported magnitude for the source
- **epoch** : epoch of RA, DEC after applying proper motions from catalog (if any)

All tables where columns have been renamed have '.meta["converted"]' set to 'True'.

Parameters

input
[str, list] Filenames of images to be aligned to astrometric catalog

catalog
[str, optional] Name of catalog to extract astrometric positions for sources in the input images' field-of-view. Default: GAIAeDR3. Only those catalogs listed in SUPPORTED_CATALOGS will be returned with standardized column names. Catalog names are case-insensitive.

output
[str, optional] Filename to give to the astrometric catalog read in from the master catalog web service. If None, no file will be written out.

gaia_only
[bool, optional] Specify whether or not to only use sources from GAIA in output catalog

existing_wcs
[stwcs.wcsutil.HSTWCS] existing WCS object specified by the user

num_sources
[int] Maximum number of brightest/faintest sources to return in catalog. If `num_sources` is negative, return that number of the faintest sources. By default, all sources are returned.

use_footprint
[bool, optional] Use the image footprint as the source identification bounds insted of using a circle centered on a given RA and Dec? By default, use_footprint = False.

full_catalog
[bool, optional] Return the full set of columns provided by the web service.

user_epoch
[str or float, optional] Epoch of returned astrometric catalog. If 'match', match the epoch of the specified observation. If None, return the values at the epoch of the catalog. If a decimal year is specified, returned values will be for that epoch.

Returns

ref_table
[astropy.table.Table] Astropy Table object of the catalog

Notes

This function will point to astrometric catalog web service defined through the use of the ASTROMETRIC_CATALOG_URL environment variable.

`drizzlepac.haputils.astrometric_utils.`**`compute_radius`**`(`*`wcs`*`)`

Compute the radius from the center to the furthest edge of the WCS.

Parameters

wcs
[stwcs.wcsutil.HSTWCS object] HSTWCS object for an exposure including a valid footprint

drizzlepac.haputils.astrometric_utils.**build_auto_kernel**(*imgarr*, *whtarr*, *fwhm=3.0*, *threshold=None*, *source_box=7*, *good_fwhm=[1.0, 4.0]*, *num_fwhm=30*, *isolation_size=11*, *saturation_limit=70000.0*, *log_level=20*)

Build kernel for use in source detection based on image PSF This algorithm looks for an isolated point-source that is non-saturated to use as a template for the source detection kernel. Failing to find any suitable sources, it will return a Gaussian2DKernel based on the provided FWHM as a default.

Parameters

imgarr
[ndarray] Image array (ndarray object) with sources to be identified

whtarr
[ndarray] Image array (ndarray object) for `imgarr` that provides an estimate of the weight or errors for the processed image.

fwhm
[float] Value of FWHM to use for creating a Gaussian2DKernel object in case no suitable source can be identified in the image.

threshold
[float] Value from the image which serves as the limit for determining sources. If None, compute a default value of (background+5*rms(background)). If threshold < 0.0, use absolute value as scaling factor for default value.

source_box
[int] Size of box (in pixels) which defines the minimum size of a valid source.

isolation_size
[int] Separation (in pixels) to use to identify sources that are isolated from any other sources in the image.

saturation_limit
[float] Flux in the image that represents the onset of saturation for a pixel.

Notes

Ideally, it would be best to determine the saturation_limit value from the data itself, perhaps by looking at the pixels flagged (in the DQ array) as saturated and selecting the value less than the minimum flux of all those pixels, or maximum pixel value in the image if non-were flagged as saturated (in the DQ array).

drizzlepac.haputils.astrometric_utils.**find_fwhm**(*psf*, *default_fwhm*)

Determine FWHM for auto-kernel PSF

This function iteratively fits a Gaussian model to the extracted PSF using photutils.psf.IterativelySubtractedPSFPhotometry to determine the FWHM of the PSF.

Parameters

psf
[ndarray] Array (preferably a slice) containing the PSF to be measured.

default_fwhm
[float] Starting guess for the FWHM

Returns

fwhm
[float] Value of the computed Gaussian FWHM for the PSF

drizzlepac.haputils.astrometric_utils.**get_catalog**(*ra*, *dec*, *sr=0.1*, *epoch=None*, *catalog='GSC241'*)

Extract reference catalog from VO web service.

Queries the catalog available at the SERVICELOCATION specified for this module to get any available astrometric source catalog entries around the specified position in the sky based on a cone-search.

Parameters

ra
[float] Right Ascension (RA) of center of field-of-view (in decimal degrees)

dec
[float] Declination (Dec) of center of field-of-view (in decimal degrees)

sr
[float, optional] Search radius (in decimal degrees) from field-of-view center to use for sources from catalog. Default: 0.1 degrees

epoch
[float, optional] Catalog positions returned for this field-of-view will have their proper motions applied to represent their positions at this date, if a value is specified at all, for catalogs with proper motions.

catalog
[str, optional] Name of catalog to query, as defined by web-service. Default: 'GSC241'

Returns

csv
[CSV object] CSV object of returned sources with all columns as provided by catalog

drizzlepac.haputils.astrometric_utils.**get_catalog_from_footprint**(*footprint*, *epoch=None*, *catalog='GSC241'*)

Extract catalog from VO web service based on the specified footprint

Queries the catalog available at the SERVICELOCATION specified for this module to get any available astrometric source catalog entries around the specified position in the sky based on the input footprint.

Parameters

footprint
[numpy.ndarray] Array of RA, Dec points that describe the footprint polygon

epoch
[float, optional] Catalog positions returned for this field-of-view will have their proper motions applied to represent their positions at this date, if a value is specified at all, for catalogs with proper motions.

catalog
[str, optional] Name of catalog to query, as defined by web-service. Default: 'GSC241'

Returns

csv
[CSV object] CSV object of returned sources with all columns as provided by catalog

drizzlepac.haputils.astrometric_utils.**extract_sources**(*img*, *dqmask=None*, *fwhm=3.0*, *kernel=None*, *photmode=None*, *segment_threshold=None*, *dao_threshold=None*, *dao_nsigma=3.0*, *source_box=7*, *classify=True*, *centering_mode='starfind'*, *nlargest=None*, *outroot=None*, *plot=False*, *vmax=None*, *deblend=False*, *log_level=20*)

Use photutils to find sources in image based on segmentation.

Parameters

img
[ndarray] Numpy array of the science extension from the observations FITS file.

dqmask
[ndarray] Bitmask which identifies whether a pixel should be used (1) in source identification or not(0). If provided, this mask will be applied to the input array prior to source identification.

fwhm
[float] Full-width half-maximum (fwhm) of the PSF in pixels.

kernel
[ndarray] A numpy array representing the PSF to be used for identifying sources. Image will be convolved with this kernel to highlight sources with shapes similar to the kernel. [Starting in v3.6.0] If None is given, a default 2D Gaussian kernel will be generated with `build_gaussian_kernel` and used based on `fwhm` and `source_box` for the size.

photmode
[str] Specification of the observation filter configuration used for the exposure as reported by the 'PHOTMODE' keyword from the PRIMARY header.

segment_threshold
[ndarray or None] Value from the image which serves as the limit for determining sources. If None, compute a default value of (background+5*rms(background)).

dao_threshold
[float, optional] [Deprecated] This parameter is not used. In fact, it now gets computed internally using the `sigma_clipped_bkg()` function which uses the `dao_nsigma` parameter.

dao_nsigma
[float] This number gets used to determine the threshold for detection of point sources. The threshold gets computed using a simple `mean + dao_nsigma * rms`, where the `mean` and `rms` are computed from the background levels using `astropy.stats.sigma_clipped_stats`.

source_box
[int] Size of each side of a box (in pixels) which defines the minimum size of a valid source.

classify
[bool] Specify whether or not to apply classification based on invarient moments of each source to determine whether or not a source is likely to be a cosmic-ray, and not include those sources in the final catalog.

centering_mode
[str] "segmentaton" or "starfind" Algorithm to use when computing the positions of the detected sources. Centering will only take place after `threshold` has been determined, and sources are identified using segmentation. Centering using `segmentation` will rely on `photutils.segmentation.SourceCatalog` to generate the properties for the source catalog. Centering using `starfind` will use `photutils.detection.IRAFStarFinder` to characterize each source in the catalog.

nlargest
[int, None] Number of largest (brightest) sources in each chip/array to measure when using 'starfind' mode.

outroot
[str, optional] If specified, write out the catalog of sources to the file with this name rootname.

plot
[bool, optional] Specify whether or not to create a plot of the sources on a view of the image.

vmax
[float, optional] If plotting the sources, scale the image to this maximum value.

deblend
[bool, optional] Specify whether or not to apply photutils deblending algorithm when evaluating each of the identified segments (sources) from the chip.

log_level

[int, optional] The desired level of verboseness in the log statements displayed on the screen and written to the .log file. Default value is 20, or 'info'.

drizzlepac.haputils.astrometric_utils.**classify_sources**(*catalog*, *fwhm*, *sources=None*)

Convert moments_central attribute for source catalog into star/cr flag.

This algorithm interprets the central_moments from the SourceCatalog generated for the sources as more-likely a star or a cosmic-ray. It is not intended or expected to be precise, merely a means of making a first cut at removing likely cosmic-rays or other artifacts.

Parameters

catalog
[photutils.segmentation.SourceCatalog] The photutils catalog for the image/chip.

sources
[tuple] Range of objects from catalog to process as a tuple of (min, max). If None (default) all sources are processed.

Returns

srctype
[ndarray] An ndarray where a value of 1 indicates a likely valid, non-cosmic-ray source, and a value of 0 indicates a likely cosmic-ray.

drizzlepac.haputils.astrometric_utils.**generate_source_catalog**(*image*, *dqname='DQ'*, *output=False*, *fwhm=3.0*, ***detector_pars*)

Build source catalogs for each chip using photutils.

The catalog returned by this function includes sources found in all chips of the input image with the positions translated to the coordinate frame defined by the reference WCS `refwcs`. The sources will be - identified using photutils segmentation-based source finding code - ignore any input pixel which has been flagged as 'bad' in the DQ array, should a DQ array be found in the input HDUList. - classified as probable cosmic-rays (if enabled) using central_moments properties of each source, with these sources being removed from the catalog.

Parameters

image
[astropy.io.fits.HDUList] Input image as an astropy.io.fits HDUList.

dqname
[str] EXTNAME for the DQ array, if present, in the input image HDUList.

output
[bool] Specify whether or not to write out a separate catalog file for all the sources found in each chip.

fwhm
[float] Full-width half-maximum (fwhm) of the PSF in pixels.

Returns

source_cats
[dict] Dict of astropy Tables identified by chip number with each table containing sources from image extension `('sci', chip)`.

drizzlepac.haputils.astrometric_utils.**generate_sky_catalog**(*image*, *refwcs*, *dqname='DQ'*, *output=False*)

Build source catalog from input image using photutils.

This script borrows heavily from build_source_catalog.

The catalog returned by this function includes sources found in all chips of the input image with the positions translated to the coordinate frame defined by the reference WCS `refwcs`. The sources will be - identified using photutils segmentation-based source finding code - ignore any input pixel which has been flagged as 'bad' in the DQ array, should a DQ array be found in the input HDUList. - classified as probable cosmic-rays (if enabled) using central_moments properties of each source, with these sources being removed from the catalog.

Parameters

image
[astropy.io.fits.HDUList] Input image.

refwcs
[stwcs.wcsutil.HSTWCS] Definition of the reference frame WCS.

dqname
[str, optional] EXTNAME for the DQ array, if present, in the input image.

output
[bool, optional] Specify whether or not to write out a separate catalog file for all the sources found in each chip.

Returns

master_cat
[`astropy.table.Table`] Source catalog for all 'valid' sources identified from all chips of the input image with positions translated to the reference WCS coordinate frame.

drizzlepac.haputils.astrometric_utils.**compute_photometry**(*catalog*, *photvals*)

Compute magnitudes for sources from catalog based on observations photmode.

Magnitudes will be AB mag values.

Parameters

catalog
[astropy.table.Table] Astropy Table with 'source_sum' column for the measured flux for each source.

photmode
[str] Specification of the observation filter configuration used for the exposure as reported by the 'PHOTMODE' keyword from the PRIMARY header.

Returns

phot_cat
[astropy.table.Table] Astropy Table object of input source catalog with added column for ABMAG photometry (in magnitudes).

drizzlepac.haputils.astrometric_utils.**filter_catalog**(*catalog*, *bright_limit=1.0*, *max_bright=None*, *min_bright=20*, *colname='vegamag'*)

Create a new catalog selected from input based on photometry.

Parameters

catalog
[astropy.table.Table] Table containing the full set of identified sources.

bright_limit
[float] Fraction of catalog based on brightness that should be retained. Value of 1.00 means full catalog.

max_bright
[int] Maximum number of sources to keep regardless of `bright_limit`.

min_bright
[int] Minimum number of sources to keep regardless of `bright_limit`.

colname
[str] Name of column to use for selection/sorting.

Returns

new_catalog
[astropy.table.Table] New table which only has the sources that meet the selection criteria.

drizzlepac.haputils.astrometric_utils.**build_self_reference**(*filename*, *clean_wcs=False*)

This function creates a reference, undistorted WCS that can be used to apply a correction to the WCS of the input file.

Parameters

filename
[str] Filename of image which will be corrected, and which will form the basis of the undistorted WCS.

clean_wcs
[bool] Specify whether or not to return the WCS object without any distortion information, or any history of the original input image. This converts the output from `utils.output_wcs()` into a pristine stwcs.wcsutil.HSTWCS object.

Returns

customwcs
[stwcs.wcsutil.HSTWCS] HSTWCS object which contains the undistorted WCS representing the entire field-of-view for the input image.

Examples

This function can be used with the following syntax to apply a shift/rot/scale change to the same image:

```
>>> import buildref
>>> from drizzlepac import updatehdr
>>> filename = "jce501erq_flc.fits"
>>> wcslin = buildref.build_self_reference(filename)
>>> updatehdr.updatewcs_with_shift(filename, wcslin, xsh=49.5694,
... ysh=19.2203, rot = 359.998, scale = 0.9999964)
```

drizzlepac.haputils.astrometric_utils.**within_footprint**(*img*, *wcsobj*, *x*, *y*)

Determine whether input x, y fall in the science area of the image.

Parameters

img
[ndarray] ndarray of image where non-science areas are marked with value of NaN.

wcsobj
[stwcs.wcsutil.HSTWCS] HSTWCS or WCS object with naxis terms defined.

x, y
[ndarray] arrays of x, y positions for sources to be checked.

Returns

mask
[ndarray] Boolean array of same length as x,y arrays where sources that fall within the footprint are True.

drizzlepac.haputils.astrometric_utils.**find_hist2d_offset**(*filename*, *reference*, *refwcs=None*, *refnames=['ra', 'dec']*, *match_tolerance=5.0*, *chip_catalog=True*, *search_radius=15.0*, *min_match=10*, *classify=True*)

Iteratively look for the best cross-match between the catalog and ref.

Parameters

filename
[astropy.io.fits.HDUList or str] Single image to extract sources for matching to the external astrometric catalog.

reference
[str or astropy.table.Table] Reference catalog, either as a filename or `astropy.Table` containing astrometrically accurate sky coordinates for astrometric standard sources.

refwcs
[stwcs.wcsutil.HSTWCS] This WCS will define the coordinate frame which will

be used to determine the offset. If None is specified, use the WCS from the input image `filename` to build this WCS using `build_self_reference()`.

refnames
[list] List of table column names for sky coordinates of astrometric standard sources from reference catalog.

match_tolerance
[float] Tolerance (in pixels) for recognizing that a source position matches an astrometric catalog position. Larger values allow for lower accuracy source positions to be compared to astrometric catalog

chip_catalog
[bool] Specify whether or not to write out individual source catalog for each chip in the image.

search_radius
[float] Maximum separation (in arcseconds) from source positions to look for valid cross-matches with reference source positions.

min_match
[int] Minimum number of cross-matches for an acceptable determination of the offset.

classify
[bool] Specify whether or not to use central_moments classification to ignore likely cosmic-rays/bad-pixels when generating the source catalog.

Returns

best_offset
[tuple] Offset in input image pixels between image source positions and astrometric catalog positions that results in largest number of matches of astrometric sources with image sources

seg_xy, ref_xy
[astropy.table.Table] Source catalog and reference catalog, respectively, used for determining the offset. Each catalog includes sources for the entire field-of-view, not just a single chip.

drizzlepac.haputils.astrometric_utils.**build_wcscat**(*image*, *group_id*, *source_catalog*)

Return a list of tweakwcs.correctors.FITSWCSCorrector objects for all chips in an image.

Parameters

image
[str, astropy.io.fits.HDUList] Either filename or HDUList of a single HST observation.

group_id
[int] Integer ID for group this image should be associated with; primarily used when separate chips are in separate files to treat them all as one exposure.

source_catalog
[dict] If provided, these catalogs will be attached as `catalog` entries in each chip's `FITSWCSCorrector` object. It should be provided as a dict of astropy Tables identified by chip number with each table containing sources from image extension `('sci', chip)` as generated by `generate_source_catalog()`.

Returns

wcs_catalogs
[list of tweakwcs.correctors.FITSWCSCorrector] List of tweakwcs.correctors.FITSWCSCorrector objects defined for all chips in input image.

drizzlepac.haputils.astrometric_utils.**compute_similarity**(*image*, *reference*)

Compute a similarity index for an image compared to a reference image.

Similarity index is based on a the general algorithm used in the AmphiIndex algorithm.

- identify slice of image that is a factor of 256 in size
- rebin image slice down to a (256,256) image
- rebin same slice from reference down to a (256,256) image
- sum the differences of the rebinned slices
- divide absolute value of difference scaled by reference slice sum

---

Note: This index will typically return values < 0.1 for similar images, and values > 1 for dis-similar images.

Parameters

image
[ndarray] Image (as ndarray) to measure

reference
[ndarray] Image which serves as the 'truth' or comparison image.

Returns

similarity_index
[float] Value of similarity index for `image`

drizzlepac.haputils.astrometric_utils.**determine_focus_index**(*img*, *sigma=1.5*)

Determine blurriness indicator for an image

This returns a single value that serves as an indication of the sharpness of the image based on the max pixel value from the image after applying a Laplacian-of-Gaussian filter with sigma.

Implementation based on discussion from: https://stackoverflow.com/questions/7765810/is-there-a-way-to-detect-if-an-image-is-blurry

as supported by: Pertuz, S., et.al., 2013, Pattern Recognition, 46:1415–1432

This index needs to be based on 'max' value in order to avoid field-dependent biases, since the 'max' value will correspond to point-source-like sources regardless of the field (nebula, galaxy, ...). Care must be taken, though, to ignore cosmic-rays as much as possible as they will mimic real sources without providing information on the actual focus through the optics. Similarly, saturation regions must also be ignored as they also only indicate a detector feature, not the focus through the optics or alignment of actual sources.

drizzlepac.haputils.astrometric_utils.**max_overlap_diff**(*total_mask*, *singlefiles*, *prodfile*, *sigma=2.0*, *scale=1*, *lsigma=3.0*)

Determines the difference in the region of max overlap for all drizzled products

Parameters

total_mask
[ndarray] Mask (array) showing where each input exposure contributes to the final drizzle product `prodfile`. This could be created using `cell_utils.SkyFootprint`.

singlefiles
[list] List of filenames for each single input exposure drizzled onto the same WCS as the final drizzle product `prodfile`

prodfile
[str] Filename for the final drizzle product

scale
[int, optional] Factor to use in downsizing (resizing smaller) the images to be evaluated. The larger the value, the less sensitive this measurement becomes.

sigma
[float, optional] Size of default kernel (in pixels) to use for determining the focus.

Returns

diff_dict
[dictionary] Dictionary of difference scores for each input exposure drizzle product (from `singlefiles`) calculated for the region of maximum overlap with the final drizzle product `prodfile`. Entries for each singlefile includes:

distance	Hamming distance of singlefile from prodfile
focus	focus index of singlefile
focus_pos	position for best focus in singlefile
product_focus	focus index for prodfile
product_focus_pos	position for best focus in prodfile

drizzlepac.haputils.astrometric_utils.**detect_point_sources**(*arr*, *background=None*, *nsigma=4*, *log_sigma=3.0*, *scale=1*, *sigma=3.0*, *exp_weight=None*)

haputils.astroquery_utils

Wrappers for astroquery-related functionality

drizzlepac.haputils.astroquery_utils.**retrieve_observation**(*obsid*, *suffix=['FLC']*, *archive=False*, *clobber=False*, *product_type=None*)

Simple interface for retrieving an observation from the MAST archive

If the input obsid is for an association, it will request all members with the specified suffixes.

Parameters

obsid
[string or list of strings] ID or list of IDs for observations to be retrieved from the MAST archive. Only the IPPSSOOT (rootname) of exposure or ASN needs to be provided; eg., ib6v06060. Additionally, a wild-carded `obsid` can be provided to retrieve all products for a visit; e.g., "ib6v06*". Data from multiple ASNs, exposures or visits can be retrieved by simply providing them as a list.

suffix
[list, optional] List containing suffixes of files which should be requested from MAST. Default value "['FLC']".

archive
[Boolean, optional] Retain copies of the downloaded files in the astroquery created sub-directories? Default is "False".

clobber
[Boolean, optional] Download and Overwrite existing files? Default is "False".

product_type
[str, optional] Specify what type of product you want from the archive, either 'pipeline' or 'HAP' or 'both' (default). By default, all versions of the products processed for the requested datasets will be returned. This would include:

- **pipeline**
  [files processed by `runastrodriz` to include the latest] distortion calibrations and the best possible alignment to GAIA with `ipppssoot_fl[tc].fits` filenames for FLT/FLC files.
- **HAP**
  [files processed as a single visit and aligned (as possible) to GAIA] with `hst_<propid>_<visit>_<instr>_<det>_<filter>_<ipppssoo>_fl[tc].fits` filenames.

Returns

local_files
[list] List of filenames

haputils.analyze

Utilities to analyze an input and determine whether the input is viable for a given process

The fundamental function analyze_data opens an input list containing FLT and/or FLC FITS filenames in order to access the primary header data. Based upon the values of specific FITS keywords, the function determines whether or not each file within this dataset can or should be reconciled against an astrometric catalog and, for multiple images, used to create a mosaic.

The functions, mvm_analyze_wrapper and analyze_wrapper, are thin wrappers around analyze_data to accommodate special case uses. The mvm_analyze_wrapper takes a filename as input and returns a boolean indicator. The analyze_wrapper has the same function signature as analyze_data, but it returns a list instead of an astropy table.

drizzlepac.haputils.analyze.**analyze_data**(*input_file_list*, *log_level=10*, *type=''*)

Determine if images within the dataset can be aligned

Parameters

input_file_list
[list] List containing FLT and/or FLC filenames for all input images which comprise an associated dataset where 'associated dataset' may be a single image, multiple images, an HST association, or a number of HST associations

log_level
[int, optional] The desired level of verboseness in the log statements displayed on the screen and written to the .log file. Default value is 20, or 'info'.

type
[string] String indicating whether this file is for MVM or some other processing. If type == "MVM", then Grism/Prism data is ignored. If type == "" (default) or any other string, the Grism/Prism data is considered available for processing unless there is some other issue (i.e., exposure time of zero).

Returns

output_table
[object] Astropy Table object containing data pertaining to the associated dataset, including the do_process bool. It is intended this table is updated by subsequent functions for bookkeeping purposes.

analyze_data_good_index
[list] indices of the viable images in the input_file_list

Notes

The keyword/value pairs below define the "cannot process categories". OBSTYPE : is not IMAGING MTFLAG : T SCAN-TYP : C or D (or !N) FILTER : G*, PR*, where G=Grism and PR=Prism FILTER1 : G*, PR*, where G=Grism and PR=Prism FILTER2 : G*, PR*, where G=Grism and PR=Prism TARGNAME : DARK, TUNGSTEN, BIAS, FLAT, EARTH-CALIB, DEUTERIUM EXPTIME : 0 CHINJECT : is not NONE DRIZCORR : OMIT, SKIPPED

The keyword/value pairs below define the category which the data can be processed, but the results may be compromised FGSLOCK : FINE/GYRO, FINE/GY, COARSE, GYROS

FITS Keywords only for WFC3 data: SCAN_TYP, FILTER, and CHINJECT (UVIS) FITS Keywords only for ACS data: FILTER1 and FILTER2

Please be aware of the FITS keyword value NONE vs the Python None.

haputils.align_utils

class drizzlepac.haputils.align_utils.**AlignmentTable**(*input_list*, *clobber=False*, *dqname='DQ'*, *process_type=''*, *log_level=0*, ***alignment_pars*)

This class handles alignment operations for HST data.

Steps managed by this class include:

- **find_alignment_sources** : iterates over all inputs and identifies sources suitable for alignment.
- **perform_fit** : cross-matches sources using user-specified method and performs fit to user-specified catalog. Cross-matching options include: `relative`, `2dhist` or `default` as reported by the `get_fit_methods` method. It then populates the table with the results for this fit.
- **select_fit** : extracts the 'best' fit to be applied to the data with the 'best' fit determined externally by the user based on the set of fit results stored in the `fit_dict` table.
- **apply_fit** : Updates all input image WCSs with the result of the selected 'best' fit

Parameters

input_list
[list] List of input file names to be provided to `analyze.analyze_data` function.

clobber
[bool, optional] Specifies whether or not to overwrite data on disk with updated versions of the data.

dqname
[str, optional] Allows the user to customize the name of the extension (`extname`) containing the data quality flags to be applied to the data during source identification.

process_type
[str, optional] Specifies what type of data processing is being done on the input data. Values include: '' (default for pipeline processing), 'SVM', 'MVM'.

log_level
[int, optional] Set the logging level for this processing

alignment_pars
[dict] Set of alignment parameters to be used for the input data. **alignment_pars needs to contain the following entries:

```
{'fwhmpsf': 0.12,  # kernel defining, source finding par
 # background computing pars
 'box_size': BKG_BOX_SIZE,
 'win_size': BKG_FILTER_SIZE,
 'bkg_estimator': SExtractorBackground,
 'rms_estimator': StdBackgroundRMS,
 'nsigma': 5.,
 'threshold_flag': None,
 # object finding pars
 'source_box': 7,
 'classify': True,
 'centering_mode': "starfind",
 'nlargest': None,
 'plot': False,
 'vmax': None,
 'deblend': False
}
```

class drizzlepac.haputils.align_utils.**HAPImage**(*filename*)

Core class defining interface for each input exposure/product

---

Note: This class is compatible with the CatalogImage class, while including additional functionality and attributes required for processing beyond catalog generation.

---

drizzlepac.haputils.align_utils.**match_relative_fit**(*imglist*, *reference_catalog*, ***fit_pars*)

Perform cross-matching and final fit using relative matching algorithm

The fitting performed by this algorithm first aligns all input images specified in the imglist to the FIRST image in that list. When this fit is successful, it insures that the images are aligned RELATIVE to each other. This set of co-aligned WCSs are then fit to the specified reference_catalog to improve the absolute astrometry for all input images (when successful).

Parameters

imglist
[list] List of input image tweakwcs.correctors.FITSWCSCorrector objects with metadata and source catalogs

reference_catalog
[Table] Astropy Table of reference sources for this field

fit_pars
[dict] Set of parameters and values to be used for the fit. This should include `fitgeom` as well as any tweakwcs.XYXYMatch parameter which the user feels needs to be adjusted to work best with the input data.

Returns

imglist
[list] List of input image tweakwcs.correctors.FITSWCSCorrector objects swith metadata and source catalogs

drizzlepac.haputils.align_utils.**match_default_fit**(*imglist*, *reference_catalog*, ***fit_pars*)

Perform cross-matching and final fit using default tolerance matching

This function performs the specified type of fit ('general', 'rscale', ...) directly between all input images and the reference catalog. If successful, each input image will have its absolute WCS aligned to the reference catalog.

Parameters

imglist
[list] List of input image tweakwcs.correctors.FITSWCSCorrector objects with metadata and source catalogs

reference_catalog
[Table] Astropy Table of reference sources for this field

fit_pars
[dict] Set of parameters and values to be used for the fit. This should include `fitgeom` as well as any tweakwcs.XYXYMatch parameter which the user feels needs to be adjusted to work best with the input data.

Returns

imglist
[list] List of input image tweakwcs.correctors.FITSWCSCorrector objects with metadata and source catalogs

drizzlepac.haputils.align_utils.**match_2dhist_fit**(*imglist*, *reference_catalog*, ***fit_pars*)

Perform cross-matching and final fit using 2dHistogram matching

This function performs cross-matching of each separate input image to the sources in the reference catalog by looking for common integer offsets between all sources in the input list and all sources in the reference catalog. This offset is then used as the starting point for the final fit to the reference catalog to align each input image SEPARATELY to the reference catalog.

Parameters

imglist
[list] List of input image tweakwcs.correctors.FITSWCSCorrector objects with metadata and source catalogs

reference_catalog
[Table] Astropy Table of reference sources for this field

fit_pars
[dict] Set of parameters and values to be used for the fit. This should include `fitgeom` as well as any tweakwcs.XYXYMatch parameter which the user feels needs to be adjusted to work best with the input data.

Returns

imglist
[list] List of input image tweakwcs.correctors.FITSWCSCorrector objects with metadata and source catalogs

HAP Parameters

Shown below are the parameters that are used by the Hubble Advanced Products. We include the parameter, the default value for WFC3 processing, and a description of that parameter.

Initialize HAP

_mdriztab_btn_: str (*default=Update From MDRIZTAB*)
Immediately read and use the values from the MDRIZTAB.

runfile: str (*default="astrodrizzle.log"*)
File for logging the processing.

wcskey: str (*default=""*)
WCS version to use in processing

proc_unit: str (*default="native"*)
Units used during processing: "native", "electrons"

coeffs: bool (*default=True*)
Use header-based distortion coefficients?

context: bool (*default=True*)
Create context image during final drizzle?

group: str (*default=""*)
Single extension or group to be combined/cleaned.

build: bool (*default=True*)
Create multi-extension output file for final drizzle?

crbit: int (*default=4096*)
Bit value for CR ident. in DQ array.

stepsize: int (*default=10*)
Step size for drizzle coordinate computation.

resetbits: int (*default="4096"*)
Bit values to reset in all input DQ arrays.

num_cores: int (*default=1*)
Max CPU cores to use (n<2 disables, None = auto-decide).

in_memory: bool (*default=False*)
Process everything in memory to minimize disk I/O?

Instrument Parameters

gnkeyword: str (*default="ATODGNA,ATODGNB,ATODGNC,ATODGND"*)
rnkeyword: str (*default="READNSEA,READNSEB,READNSEC,READNSED"*)
Keyword used to specify a value, which is used to override the instrument specific default readnoise values. The value is assumed to be in units of electrons. This parameter should not be populated if the `rdnoise` parameter is in use.

expkeyword: str (*default="EXPTIME"*)
Keyword used to specify a value, which is used to override the default exposure time image header values. The value is assumed to be in units of seconds. This parameter should not be populated if the `exptime` parameter is in use.

gain: str (*default=""*)
Value used to override instrument specific default gain values. The value is assumed to be in units of electrons/count. This parameter should not be populated if the `gainkeyword` parameter is in use.

rdnoise: str (*default=""*)
Value used to override instrument specific default readnoise values. The value is assumed to be in units of electrons. This parameter should not be populated if the `rnkeyword` parameter is in use.

exptime: str (*default=""*)

State of input files

restore: bool (*default=False*)
Copy input files FROM archive directory for processing?

preserve: bool (*default=False*)
Copy input files to archive directory, if not already archived?

overwrite: bool (*default=False*)
Copy input files into archive, overwriting if required?

clean: bool (*default=True*)
Delete temporary files after completion?

Step 1: Static mask

static: bool (*default=True*)
Create static bad-pixel mask from the data?

static_sig: float (*default=4.0*)
Sigma*rms below mode to clip for static mask

Step 2: Sky Subtraction

skysub: bool (*default=False*)
Turn on or off sky subtraction on the input data. When `skysub` is set to `no`, then `skyuser` field will be enabled and if user specifies a header keyword showing the sky value in the image, then that value will be used for CR-rejection but it will not be subtracted from the (drizzled) image data. If user sets `skysub` to `yes` then `skyuser` field will be disabled (and if it is not empty - it will be ignored) and user can use one of the methods available through the `skymethod` parameter to compute the sky or provide a file (see `skyfile` parameter) with values that should be subtracted from (single) drizzled images.

skymethod: str (*default="match"*)
Sky computation method: "globalmin+match","localmin", "globalmin", "match". See astrodrizzle.help for more details.

skystat: str (*default="median"*)
Statistical method for determining the sky value from the image pixel values: "median","mode","mean".

skywidth: float (*default=0.1*)
Bin width of histogram for sampling sky statistics (in sigma)

skylower: float (*default=-100.0*)
Lower limit of usable data for sky (always in electrons)

sky_bits: int (*default="16"*)
nteger mask bit values considered good pixels in DQ array

skyupper: int or null (*default=null*)
Upper limit of usable data for sky (always in electrons)

skyclip: int (*default=5*)
Number of clipping iterations

skylsigma: float (*default=4.0*)
Lower side clipping factor (in sigma)

skyusigma: float (*default=4.0*)
Upper side clipping factor (in sigma)

skymask_cat: str (*default=""*)
Catalog file listing image masks

use_static: bool (*default=True*)
Use static mask for skymatch computations?

skyfile: str (*default""*)
Name of file with user-computed sky values to be subtracted

skyuser: str (*default""*)
KEYWORD indicating a sky subtraction value if done by user

Step 3: Drizzle Separate images

driz_separate
[bool (*default=False*)] This parameter specifies whether or not to drizzle each input image onto separate output images. The separate output images will all have the same WCS as the final combined output frame. These images are used to create the median image, needed for cosmic ray rejection.

driz_sep_bits: int (*default="16"*)
Integer sum of all the DQ bit values from the input image's DQ array that should be considered 'good' when building the weighting mask. This can also be used to reset pixels to good if they had been flagged as cosmic rays during a previous run of `AstroDrizzle`, by adding the value 4096 for ACS and WFPC2 data. Please see the section on Selecting the `Bits` Parameter for a more detailed discussion.

driz_sep_kernel: str (*default="turbo"*)
Used for the initial separate drizzling operation only, this parameter specifies the form of the kernel function used to distribute flux onto the separate output images. The current options are: 'square', 'point', 'turbo', 'gaussian', and 'lanczos3'. The latter two options ('gaussian' and 'lanczos3') are not guaranteed to conserve flux, but may produce reasonable results; understand the effects of these kernels before using them. A former option 'tophat' has been removed as it was found to produce poor results. See adrizzle.help for more details.

driz_sep_wt_scl: float (*default=exposure time (from image header)*)
This parameter specifies the weighting factor for input image. If `driz_sep_wt_scl=exptime`, then the scaling value will be set equal to the exposure time found in the image header. The use of the default value is recommended for producing optimal behavior for most scenarious. It is possible to set `wt_scl`='expsq' for weighting by the square of the exposure time, which is optimal for read-noise dominated images.

driz_sep_pixfrac: float (*default=1.0*)
Fraction by which input pixels are "shrunk" before being drizzled onto the output image grid, given as a real number between 0 and 1. This specifies the size of the footprint, or "dropsize", of a pixel in units of the input pixel size. If `pixfrac` is set to less than 0.001, the kernel parameter will be reset to 'point' for more efficient processing. In the step of drizzling each input image onto a separate output image, the default value of 1.0 is best in order to ensure that each output drizzled image is fully populated with pixels from the input image. For more information, see the help for the `drizzle` task.

driz_sep_fillval: int or INDEF (*default = null*)
Value to be assigned to output pixels that have zero weight, or that receive flux from any input pixels during drizzling. This parameter corresponds to the `fillval` parameter of the `drizzle` task. If the default of `INDEF` is used, and if the weight in both the input and output images for a given pixel are zero, then the output pixel will be set to the value it would have had if the input had a non-zero weight. Otherwise, if a numerical value is provided (e.g. 0), then these pixels will be set to that value.

driz_sep_compress: bool (*default=False*)
Whether to use compression when writing out product.

Step 3a: Custom WCS for Separate Outputs

driz_sep_wcs: bool (*default=False*)
Define custom WCS for seperate output images?

driz_sep_refimage: str (*default=""*)
Reference image from which a WCS solution can be obtained.

driz_sep_rot
[float or null (*default=null*)] Position Angle of output image's Y-axis relative to North. A value of 0.0 would orient the final output image to be North up. The default of `INDEF` specifies that the images will not be rotated, but will instead be drizzled in the default orientation for the camera with the x and y axes of the drizzled image corresponding approximately to the detector axes. This conserves disk space, as these single drizzled images are only used in the intermediate step of creating a median image.

driz_sep_scale
[float or null (*default=null*)] Linear size of the output pixels in arcseconds/pixel for each separate drizzled image (used in creating the median for cosmic ray rejection). The default value of `INDEF` specifies that the undistorted pixel scale for the first input image will be used as the pixel scale for all the output images.

driz_sep_outnx
[int or null (*default=null*)] Size, in pixels, of the X axis in the output images that each input will be drizzled onto. If no value is specified, the smallest size that can accommodate the full dithered field will be used.

driz_sep_outny
[int or null (*default=null*)] Size, in pixels, of the Y axis in the output images that each input will be drizzled onto. If no value is specified, the smallest size that can accommodate the full dithered field will be used.

driz_sep_ra
[float or null (*default=null*)] Right ascension (in decimal degrees) specifying the center of the output image. If this value is not designated, the center will automatically be calculated based on the distribution of image dither positions.

driz_sep_dec
[float or null (*default=null*)] Declination (in decimal degrees) specifying the center of the output image. If this value is not designated, the center will automatically be calculated based on the distribution of image dither positions.

driz_sep_crpix1: float or null (*default=null*)
Reference pixel X position on output (CRPIX1).

driz_sep_crpix2: float or null (*default=null*)
Reference pixel Y position on output (CRPIX2).

Step 4: Create Median Image

median: bool (*default=False*)
Create a median image?

median_newmasks: bool (*default=True*)
Create new masks when doing the median?

combine_type: str (*default="minmed"*)
Type of combine operation. "minmed","iminmed","median","mean","imedian","imean","sum".

combine_nlow: int (*default=0*)
Minmxa, number of low pixels to reject.

combine_nhigh: int (*default=1*)
Minmxa, number of high pixels to reject.

combine_maskpt: float (*default=0.3*)
Percentage of weight image value below which it is flagged as a bad pixel.

combine_nsigma: str (*default="4 3"*)
Significance for accepting minimum instead of median.

combine_lthresh: ??? (*default=null*)
Lower threshold for clipping input pixel values.

combine_hthresh: ??? (*default=null*)
Upper threshold for clipping input pixel values.

combine_grow: int (*default=1*)
Radius (pixels) for neighbor rejection.

combine_bufsize: ??? (*default=null*)
Size of buffer(in Mb) for each input image.

Step 5: Blot back the median image

blot: bool (*default=False*)
Blot the median back to the input frame?

blot_interp: str (*default="poly5"*)
Interpolant (nearest,linear,poly3,poly5,sinc)

blot_sinscl: float (*default=1.0*)
Scale for sinc interpolation kernel

blot_addsky: bool (*default=True*)
Add sky using MDRIZSKY value from header?

blot_skyval: float (*default=0.0*)
Custom sky value to be added to blot image

Step 6: Remove cosmic rays with deriv, driz_cr

driz_cr: bool (*default=False*)
Perform CR rejection with deriv and driz_cr?

driz_cr_snr: str (*default="5.0 4.0"*)
Driz_cr.SNR parameter*

driz_cr_grow: int (*default=1*)
Driz_cr_grow parameter

driz_cr_ctegrow: int (*default=0*)
Driz_cr_ctegrow parameter

driz_cr_scale: str (*default="3.0 2.4"*)
Driz_cr.scale parameter

driz_cr_corr: bool (*default=False*)
Create CR cleaned _crclean file and a _crmask file?

Step 7: Drizzle final combined image

driz_combine: bool (*default=True*)
Perform final drizzle image combination?

final_pixfrac: float (*default=1.0*)
Linear size of drop in input pixels

final_fillval: int (*default=null*)
Value to be assigned to undefined output points

final_bits: str (*default="65535"*)
Integer mask bit values considered good

final_maskval: ??? (*default=null*)
Value to be assigned to regions outside SCI image

final_wht_type: str (*default="EXP"*)
Type of weighting for final drizzle

final_kernel: str (*default="square"*)
Shape of kernel function

final_wt_scl: str (*default="exptime"*)
Weighting factor for input data image

final_units: str (*default="cps"*)
Units for final drizzle image (counts or cps)

Step 7a: Custom WCS for final output

final_wcs: bool (*default=True*)
Define custom WCS for final output image?

final_rot: float (*default=0.0*)
Position Angle of drizzled image's Y-axis w.r.t. North (degrees)

final_refimage: str (*default=""*)
Reference image from which to obtain a WCS

final_scale: int (*default=null*)
Absolute size of output pixels in arcsec/pixel

final_outnx: int (*default=null*)
Size of FINAL output frame X-axis (pixels)

final_outny: int (*default=null*)
Size of FINAL output frame Y-axis (pixels)

final_ra: float (*default=null*)
right ascension output frame center in decimal degrees

final_dec: float (*default=null*)
declination output frame center in decimal degrees

final_crpix1: ??? (*default=null*)
Reference pixel X position on output (CRPIX1)

final_crpix2: ??? (*default=null*)
Reference pixel Y position on output (CRPIX2)

API for runsinglehap

The task `runsinglehap` serves as the primary interface for processing data from a single-visit into a uniform set of images. runsinglehap.py - Module to control processing of single-visit mosaics

License
License

USAGE:

```
>>> runsinglehap [-cdl] inputFilename
```

- The '-c' option allows the user to specify a customized configuration JSON file which has been tuned for specialized processing. This file should contain ALL the input parameters necessary for processing. If not specified, default configuration values will be used.
- The '-d' option will run this task in a more verbose diagnostic mode producing additional log messages will be displayed and additional files will be created.
- The '-l' option allows the user to set the desired level of verboseness in the log statements displayed on the screen and written to the .log file. Specifying "critical" will only record/display

"critical" log statements, and specifying "error" will record/display both "error" and "critical" log statements, and so on. Valid inputs: 'critical', 'error', 'warning', 'info', or 'debug'.

Python USAGE:

```
>>> python
>>> from drizzlepac import runsinglehap
>>> runsinglehap.perform(inputFilename, debug=False, input_custom_pars_
↪file=None, log_level='info')
```

drizzlepac.runsinglehap.**perform**(*input_filename*, ***kwargs*)

Main calling subroutine for the runsinglehap task.

Parameters

input_filename
[str] Name of the input csv file containing information about the files to be processed

debug
[bool, optional] display all tracebacks, and debug information? If not specified, the default value is Boolean 'False'.

input_custom_pars_file
[str, optional] Represents a fully specified input filename of a configuration JSON file which has been customized for specialized processing. This file should contain ALL the input parameters necessary for processing. If not specified, default configuration parameter values will be used.

log_level
[str, optional] The desired level of verboseness in the log statements displayed on the screen and written to the .log file. Valid inputs: 'critical', 'error', 'warning', 'info', or 'debug'. If not specified, the default value is 'info'.

This script defines the HST Advanced Products (HAP) generation portion of the calibration pipeline. This portion of the pipeline produces mosaic and catalog products. This script provides similar functionality as compared to the Hubble Legacy Archive (HLA) pipeline in that it acts as the controller for the overall sequence of processing.

Note regarding logging... During instantiation of the log, the logging level is set to NOTSET which essentially means all message levels (debug, info, etc.) will be passed from the logger to the underlying handlers, and the handlers will dispatch all messages to the associated streams. When the command line option of setting the logging level is invoked, the logger basically filters which messages are passed on to the handlers according the level chosen. The logger is acting as a gate on the messages which are allowed to be passed to the handlers.

Creation of source catalogs can be controlled through the use of environment variables:

- SVM_CATALOG_HRC
- SVM_CATALOG_SBC
- SVM_CATALOG_WFC

- SVM_CATALOG_UVIS
- SVM_CATALOG_IR

These variables can be defined using values of:

- 'on', 'true', 'yes' : Create catalogs
- 'off', 'false', 'no' : Turn off generation of catalogs

The output products can be evaluated to determine the quality of the alignment and output data through the use of the environment variable:

- SVM_QUALITY_TESTING : Turn on quality assessment processing. This environment variable, if found with an affirmative value, will turn on processing to generate a JSON file which contains the results of evaluating the quality of the generated products.

NOTE: Step 9 compares the output HAP products to the Hubble Legacy Archive (HLA) products. In order for step 9 (run_sourcelist_comparison()) to run, the following environment variables need to be set: - HLA_CLASSIC_BASEPATH - HLA_BUILD_VER

Alternatively, if the HLA classic path is unavailable, The comparison can be run using locally stored HLA classic files. The relevant HLA classic imagery and sourcelist files must be placed in a subdirectory of the current working directory called 'hla_classic'.

drizzlepac.hapsequencer.**run_hap_processing**(*input_filename*, *diagnostic_mode=False*, *input_custom_pars_file=None*, *output_custom_pars_file=None*, *phot_mode='both'*, *log_level=20*)

Run the HST Advanced Products (HAP) generation code. This routine is the sequencer or controller which invokes the high-level functionality to process the single visit data.

Parameters

input_filename
[str] The 'poller file' where each line contains information regarding an exposures taken during a single visit.

diagnostic_mode
[bool, optional] Allows printing of additional diagnostic information to the log. Also, can turn on creation and use of pickled information.

input_custom_pars_file
[str, optional] Represents a fully specified input filename of a configuration JSON file which has been customized for specialized processing. This file should contain ALL the input parameters necessary for processing. If not specified, default configuration parameter values will be used. The default is None.

output_custom_pars_file
[str, optional] Fully specified output filename which contains all the configuration parameters available during the processing session. The default is None.

phot_mode
[str, optional] Which algorithm should be used to generate the sourcelists? 'aper-

ture' for aperture photometry; 'segment' for segment map photometry; 'both' for both 'segment' and 'aperture'. Default value is 'both'.

log_level
[int, optional] The desired level of verboseness in the log statements displayed on the screen and written to the .log file. Default value is 20, or 'info'.

Returns

return_value: int
A return exit code used by the calling Condor/OWL workflow code: 0 (zero) for success, 1 for error

drizzlepac.hapsequencer.**create_catalog_products**(*total_obj_list*, *log_level*, *diagnostic_mode=False*, *phot_mode='both'*, *catalog_switches=None*)

This subroutine utilizes haputils/catalog_utils module to produce photometric sourcelists for the specified total drizzle product and it's associated child filter products.

Parameters

total_obj_list
[*TotalProduct*] total drizzle product that will be processed by catalog_utils. catalog_utils will also create photometric sourcelists for the child filter products of this total product.

log_level
[int, optional] The desired level of verboseness in the log statements displayed on the screen and written to the .log file.

diagnostic_mode
[bool, optional] generate ds9 region file counterparts to the photometric sourcelists? Default value is False.

phot_mode
[str, optional] Which algorithm should be used to generate the sourcelists? 'aperture' for aperture photometry; 'segment' for segment map photometry; 'both' for both 'segment' and 'aperture'. Default value is 'both'.

catalog_switches
[dict, optional] Specify which, if any, catalogs should be generated at all, based on detector. This dictionary needs to contain values for all instruments; namely:

SVM_CATALOG_HRC, SVM_CATALOG_SBC, SVM_CATALOG_WFC, SVM_CATALOG_UVIS, SVM_CATALOG_IR, SVM_CATALOG_PC

These variables can be defined with values of 'on'/'off'/'yes'/'no'/'true'/'false'.

Returns

product_list
[list] list of all catalogs generated.

drizzlepac.hapsequencer.**create_drizzle_products**(*total_obj_list*)

Run astrodrizzle to produce products specified in the total_obj_list.

Parameters

total_obj_list
[list] List of TotalProduct objects, one object per instrument/detector combination is a visit. The TotalProduct objects are comprised of FilterProduct and ExposureProduct objects.

Returns

product_list
[list] A list of output products

Supporting code

These modules and functions provide the core functionality for the single-visit processing.

drizzlepac.haputils.product

Definition of Super and Subclasses for the mosaic output image_list

Classes which define the total ("white light" image), filter, and exposure drizzle products.

These products represent different levels of processing with the levels noted in the 'HAPLEVEL' keyword. The 'HAPLEVEL' values are:

- 1 : calibrated (FLT/FLC) input images and exposure level drizzle products with improved astrometry
- 2 : filter and total products combined using the improved astrometry, consistent pixel scale, and oriented to North
- 3 : unused
- 4 : multi-visit mosaics aligned to a common tangent plane

class drizzlepac.haputils.product.**HAPProduct**(*prop_id*, *obset_id*, *instrument*, *detector*, *aperture_from_poller*, *filename*, *filetype*, *log_level*)

HAPProduct is the base class for the various products generated during the astrometry update and mosaicing of image data.

class drizzlepac.haputils.product.**TotalProduct**(*prop_id*, *obset_id*, *instrument*, *detector*, *aperture_from_poller*, *filename*, *filetype*, *log_level*)

A Total Detection Product is a 'white' light mosaic comprised of images acquired with one instrument, one detector, all filters, and all exposure times. The exposure time characteristic may change - TBD.

The "tdp" is short hand for TotalProduct.

class drizzlepac.haputils.product.**FilterProduct**(*prop_id*, *obset_id*, *instrument*, *detector*, *aperture_from_poller*, *filename*, *filters*, *filetype*, *log_level*)

A Filter Detection Product is a mosaic comprised of images acquired during a single visit with one instrument, one detector, a single filter, and all exposure times. The exposure time characteristic may change - TBD.

The "fdp" is short hand for FilterProduct.

class drizzlepac.haputils.product.**ExposureProduct**(*prop_id*, *obset_id*, *instrument*, *detector*, *aperture_from_poller*, *filename*, *filters*, *filetype*, *log_level*)

An Exposure Product is an individual exposure/image (flt/flc).

The "edp" is short hand for ExposureProduct.

drizzlepac.haputils.poller_utils

Utilities to interpret the pipeline poller obset information and generate product filenames

The function, interpret_obset_input, parses the file generated by the pipeline poller, and produces a tree listing of the output products. The function, parse_obset_tree, converts the tree into product catagories.

drizzlepac.haputils.poller_utils.**interpret_obset_input**(*results: str*, *log_level*)

parses the file generated by the pipeline poller, and produces a tree listing of the output products.

Parameters

results
[str or list] Input poller file name or Python list of dataset names to be processed as a single visit. Dataset names have to be either the filename of a singleton (non-associated exposure) or the name of an ASN file (e.g., jabc01010_asn.fits).

log_level
[int] The desired level of verboseness in the log statements displayed on the screen and written to the .log file.

Returns

obset_dict
[dict] Dictionary of obset objects

tdp_list
[list] list of total detection products

Notes

Interpret the database query for a given obset to prepare the returned values for use in generating the names of all the expected output products.

Input will have formated rows of one of the following three options

1) Full poller file

```
ib4606c5q_flc.fits,11665,B46,06,1.0,F555W,UVIS,/foo/bar/ib4606c5q_flc.fits
```

2) Simpler poller file (list); other info taken from file header keywords

```
ib4606c5q_flc.fits
```

3) **For updating the WFPC2 SVM aperture keyword using the poller file; it** is important that there are no spaces within the poller aperture keyword(s)

```
ib4606c5q_flc.fits, PC1-FIX;F160BN15
```

Full poller files contain filename, proposal_id, program_id, obset_id, exptime, filters, detector, pathname.

The output dict will have format (as needed by further code for creating the product filenames) of

```
obs_info_dict["single exposure product 00": {"info": '11665 06 wfc3 uvis␣
↪empty_aperture ib4606c5q f555w drc',
                                              "files": ['ib4606c5q_flc.fits
↪']}
.
.
.
obs_info_dict["single exposure product 08": {"info": '11665 06 wfc3 ir␣
↪empty_aperture ib4606clq f110w drz',
                                              "files": ['ib4606clq_flt.fits
↪']}

obs_info_dict["filter product 00": {"info": '11665 06 wfc3 uvis empty_
↪aperture ib4606c5q f555w drc',
                                     "files": ['ib4606c5q_flc.fits',
↪'ib4606c6q_flc.fits']},
.
.
.
obs_info_dict["filter product 01": {"info": '11665 06 wfc3 ir empty_
↪aperture ib4606cmq f160w drz',
                                     "files": ['ib4606cmq_flt.fits',
```

(continues on next page)

(continued from previous page)

```
↪'ib4606crq_flt.fits']},

obs_info_dict["total detection product 00": {"info": '11665 06 wfc3 uvis␣
↪empty_aperture ib4606c5q f555w drc',
                                            "files": ['ib4606c5q_flc.fits',
↪ 'ib4606c6q_flc.fits']}
.
.
.
obs_info_dict["total detection product 01": {"info": '11665 06 wfc3 ir␣
↪empty_aperture ib4606cmq f160w drz',
                                            "files": ['ib4606cmq_flt.fits',
↪ 'ib4606crq_flt.fits']}
```

The aperture keyword, which has a default value of 'empty_aperture' is filled in the case of WFPC2 observations, and the use of the two-column format.

drizzlepac.haputils.poller_utils.**parse_obset_tree**(*det_tree*, *log_level*)

Convert tree into products

Tree generated by `create_row_info()` will always have the following levels:

- detector
  - filters
    - exposure

Each exposure will have an entry dict with keys 'info' and 'filename'.

Products created will be:

- total detection product per detector
- filter products per detector
- single exposure product
- grism/prism single exposure product [only if Grism/Prism exposures]

Grism/Prism images were added to the "doProcess" list after most of this code was developed as they need to have the same Primary WCS as the direct images from the same detector in the visit. These images are deliberately not added to the output obset_products as they will not be drizzled. They are added as individual GrismExposureProduct objects to the TotalProduct only. The GrismExposure-Product list in the TotalProduct is separate and distinct from the ExposureProduct list of direct images.

drizzlepac.haputils.poller_utils.**build_obset_tree**(*obset_table*)

Convert obset table into a tree listing all products to be created.

drizzlepac.haputils.poller_utils.**build_poller_table**(*input: str*, *log_level*, *all_mvm_exposures=[]*, *poller_type='svm'*, *include_small=True*, *only_cte=False*)

Create a poller file from dataset names for either SMV or MVM processing. Information is either gathered from the poller file or by using the filename to open the file and pulling information from the header keywords. The code treats WFPC2 differently, by uses both approaches. For WFPC2, We use simple poller files with a second column that includes the aperture. The code gathers the rest of the relevant informaiton from the header keywords.

Parameters

input
[str, list] Filename with list of dataset names, or just a Python list of dataset names, provided by the user.

log_level
[int] The desired level of verboseness in the log statements displayed on the screen and written to the . log file.

all_mvm_exposures
[str, list] List of all the MVM FLT/FLC exposure filenames in the list. This will include the filenames which are dropped because they have been processed previously.

poller_type
[str, optional] The type of poller file being processed. Either 'svm' for single visit mosaic, or 'mvm' for multi-visit mosaic. Unless explicitly specified, the default value is 'svm'.

Returns

poller_table
[Table] Astropy table object with the same columns as a poller file.

drizzlepac.haputils.catalog_utils

This script contains code to support creation of photometric sourcelists using two techniques: aperture photometry and segmentation-map based photometry.

class drizzlepac.haputils.catalog_utils.**CatalogImage**(*filename*, *num_images_mask*, *log_level*)

class drizzlepac.haputils.catalog_utils.**HAPCatalogs**(*fitsfile*, *param_dict*, *param_dict_qc*, *num_images_mask*, *log_level*, *diagnostic_mode=False*, *types=None*, *tp_sources=None*)

Generate photometric sourcelist for specified TOTAL or FILTER product image.

class drizzlepac.haputils.catalog_utils.**HAPCatalogBase**(*image*, *param_dict*, *param_dict_qc*, *diagnostic_mode*, *tp_sources*)

Virtual class used to define API for all catalogs

class drizzlepac.haputils.catalog_utils.**HAPPointCatalog**(*image*, *param_dict*, *param_dict_qc*, *diagnostic_mode*, *tp_sources*)

Generate photometric sourcelist(s) for specified image(s) using aperture photometry of point sources.

class drizzlepac.haputils.catalog_utils.**HAPSegmentCatalog**(*image*, *param_dict*, *param_dict_qc*, *diagnostic_mode*, *tp_sources*)

Generate a sourcelist for a specified image by detecting both point and extended sources using the image segmentation process.

Parameters

image
[CatalogImage object] The white light (aka total detection) or filter drizzled image

param_dict
[dictionary] Configuration values for catalog generation based upon input JSON files

diagnostic_mode
[bool] Specifies whether or not to generate the regions file used for ds9 overlay

tp_sources: dictionary
Dictionary containing computed information for each catalog type

drizzlepac.haputils.photometry_tools

Tools for aperture photometry with non native bg/error methods

This function serves to ease the computation of photometric magnitudes and errors using PhotUtils by replicating DAOPHOT's photometry and error methods. The formula for DAOPHOT's error is:

err = sqrt (Poisson_noise / epadu + area * stdev**2 + area**2 * stdev**2 / nsky)

Which gives a magnitude error:

mag_err = 1.0857 * err / flux

Where epadu is electrons per ADU (gain), area is the photometric aperture area, stdev is the uncertainty in the sky measurement and nsky is the sky annulus area. To get the uncertainty in the sky we must use a custom background tool, which also enables computation of the mean and median of the sky as well (more robust statistics). All the stats are sigma clipped. These are calculated by the functions in aperture_stats_tbl.

Note: Currently, the background computations will fully include a pixel that has ANY overlap with the background aperture (the annulus). This is to simplify the computation of the median, as a weighted median is nontrivial, and slower. Copied from https://grit.stsci.edu/HLA/software/blob/master/HLApipeline/HLApipe/scripts/photometry_tools.py

Authors

- Varun Bajaj, January 2018

Use

```
from photometry_tools import iraf_style_photometry
phot_aps = CircularAperture((sources['xcentroid'], sources['ycentroid']),r=10.)
bg_aps = CircularAnnulus((sources['xcentroid'], sources['ycentroid']), r_in=13.,␣
↪r_out=15.)
```

Simplest call:

```
photometry_tbl = iraf_style_photometry(phot_aps, bg_aps, data)
```

Pass in pixelwise error and set background to mean

```
photometry_tbl = iraf_style_photometry(phot_aps, bg_aps, data, error_array=data_
↪err, bg_method='mean')
```

Can also set the gain (if image units are DN)

```
photometry_tbl = iraf_style_photometry(phot_aps, bg_aps, data, epadu=2.5)
```

Classes and Functions

drizzlepac.haputils.photometry_tools.**iraf_style_photometry**(*phot_apertures*, *bg_apertures*, *data*, *photflam*, *photplam*, *error_array=None*, *bg_method='mode'*, *epadu=1.0*)

Computes photometry with PhotUtils apertures, with IRAF formulae

Parameters

phot_apertures
[photutils PixelAperture object (or subclass)] The PhotUtils apertures object to compute the photometry. i.e. the object returned via CirularAperture.

bg_apertures
[photutils PixelAperture object (or subclass)] The photutils aperture object to measure the background in. i.e. the object returned via CircularAnnulus.

data
[array] The data for the image to be measured.

photflam
[float] inverse sensitivity, in ergs/cm2/angstrom/electron

photplam
[float] Pivot wavelength, in angstroms

error_array
[array] (Optional) The array of pixelwise error of the data. If none, the Poisson noise term in the error computation will just be the square root of the flux/epadu. If not none, the aperture_sum_err column output by aperture_photometry (divided by epadu) will be used as the Poisson noise term.

bg_method: string
{'mean', 'median', 'mode'}, optional. The statistic used to calculate the background. All measurements are sigma clipped. Default value is 'mode'. NOTE: From DAOPHOT, mode = 3 * median - 2 * mean.

epadu
[float] (optional) Gain in electrons per adu (only use if image units aren't e-). Default value is 1.0

Returns

An astropy Table with columns as follows:
X-Center Y-Center RA DEC ID MagAp1 MagErrAp1 MagAp2 MagErrAp2 MSkyAp2 StdevAp2 FluxAp2 CI Flags

drizzlepac.haputils.photometry_tools.**compute_phot_error**(*flux_variance*, *bg_phot*, *bg_method*, *ap_area*, *epadu=1.0*)

Computes the flux errors using the DAOPHOT style computation. Originally taken from https://github.com/spacetelescope/wfc3_photometry/blob/527815bf580d0361754281b608008e539ed84368/photometry_tools/photometry_with_errors.py#L137

Parameters

flux_variance
[array] flux values

bg_phot
[array] background brightness values.

bg_method
[string] background method

ap_area
[array] the area of the aperture in square pixels

epadu
[float] (optional) Gain in electrons per adu (only use if image units aren't e-). Default value is 1.0

Returns

flux_error
[array] an array of flux errors

drizzlepac.haputils.photometry_tools.**convert_flux_to_abmag**(*in_flux*, *photflam*, *photplam*)
converts flux (in units of electrons/sec) to ABMAG

Parameters

in_flux
[astropy.table.column.Column object] flux values to convert to ABMAG, in electrons/second

photflam
[float] inverse sensitivity, in ergs/cm2/angstrom/electron

photplam
[float] pivot wavelength, in angstroms

Returns

abmag
[astropy.table.column.Column object] input flux values converted to ABMAG

Associated Helper Code

These modules and functions assist with some of the logistics associated with single-visit processing.

drizzlepac.haputils.processing_utils

Utilities to support creation of Hubble Advanced Pipeline(HAP) products.

drizzlepac.haputils.processing_utils.**refine_product_headers**(*product*, *total_obj_list*)
Refines output product headers to include values not available to AstroDrizzle.

A few header keywords need to have values computed to reflect the type of product being generated, which in some cases can only be done using information passed in from the calling routine. This function insures that all header keywords have been populated with values appropriate for the type of product being processed.

Parameters

product
[str or object] Filename or HDUList object for product to be updated

total_obj_list: list
List of TotalProduct objects which are composed of Filter and Exposure Product objects

drizzlepac.haputils.processing_utils.**compute_sregion**(*image*, *extname='SCI'*)
Compute the S_REGION keyword for a given WCS.

Parameters

image
[Astropy io.fits HDUList object] Image to update with the S_REGION keyword in each of the SCI extensions.

extname
[str, optional] EXTNAME value for extension containing the WCS(s) to be updated

Python Interface for MVM Processing Code

The *hapmultisequencer* module is part of the drizzlepac package and calls utilities which are intended to be used with Multi-visit Mosaic (MVM) data and to be imported into other Python programs or a Python session for interactive data analysis.

hapmultisequencer

This module provides the primary user interface for performing MVM processing. This script defines the HST Advanced Products (HAP) Multi-visit (MVM) generation portion of the calibration pipeline. This portion of the pipeline produces mosaic products. This script provides similar functionality as compared to the Hubble Legacy Archive (HLA) pipeline in that it provides the overall sequence of the processing.

Note regarding logging... During instantiation of the log, the logging level is set to NOTSET which essentially means all message levels (debug, info, etc.) will be passed from the logger to the underlying handlers, and the handlers will dispatch all messages to the associated streams. When the command line option of setting the logging level is invoked, the logger basically filters which messages are passed on to the handlers according the level chosen. The logger is acting as a gate on the messages which are allowed to be passed to the handlers.

Some environment variables can be used to specify whether or not to include some types of data when creating MVM products. By default, the code will generate SkyCell layers from all data specified by the user. These variables allow the user to ignore types of data from the input during MVM processing.

- **MVM_INCLUDE_SMALL** : Generate MVM SkyCell layers from ACS/HRC and ACS/SBC data. These exposures typically only cover a miniscule fraction of a SkyCell and the plate scale of the SkyCell would result in a degraded representation of the original ACS/HRC and ACS/SBC data. Therefore, this variable can be set to 'off' or 'false' to turn off generation of layers from ACS/HRC and ACS/SBC data.
- **MVM_ONLY_CTE** : Generate MVM SkyCell layers using only CTE-corrected input files. If 'off' or 'false', then both CTE-corrected and non-CTE-corrected data may end up in the same SkyCell layer (image).

The output products can be evaluated to determine the quality of the alignment and output data through the use of the environment variable:

- **MVM_QUALITY_TESTING** : Turn on quality assessment processing. This environment variable, if found with an affirmative value, will turn on processing to generate a JSON file which contains the results of evaluating the quality of the generated products.

drizzlepac.hapmultisequencer.**run_mvm_processing**(*input_filename*, *skip_gaia_alignment=True*, *diagnostic_mode=False*, *use_defaults_configs=True*, *input_custom_pars_file=None*, *output_custom_pars_file=None*, *phot_mode='both'*, *custom_limits=None*, *output_file_prefix=None*, *log_level=20*)

Run the HST Advanced Products (HAP) generation code. This routine is the sequencer or controller which invokes the high-level functionality to process the multi-visit data.

Parameters

input_filename: string
The 'poller file' where each line contains information regarding an exposures considered part of the multi-visit.

skip_gaia_alignment
[bool, optional] Skip alignment of all input images to known Gaia/HSC sources in the input image footprint? If set to 'True', the existing input image alignment solution will be used instead. The default is False.

diagnostic_mode
[bool, optional] Allows printing of additional diagnostic information to the log. Also, can turn on creation and use of pickled information.

use_defaults_configs: bool, optional
If True, use the configuration parameters in the 'default' portion of the configuration JSON files. If False, use the configuration parameters in the "parameters" portion of the file. The default is True.

input_custom_pars_file: string, optional
Represents a fully specified input filename of a configuration JSON file which has been customized for specialized processing. This file should contain ALL the input parameters necessary for processing. If there is a filename present for this parameter, the 'use_defaults_configs' parameter is ignored. The default is None.

output_custom_pars_file: string, optional
Fully specified output filename which contains all the configuration parameters available during the processing session. The default is None.

phot_mode
[str, optional] Which algorithm should be used to generate the sourcelists? 'aperture' for aperture (point) photometry; 'segment' for isophotal photometry; 'both' for both 'segment' and 'aperture'. Default value is 'both'.

custom_limits
[list, optional] 4-element list containing the mosaic bounding rectangle X min and max and Y min and max values for custom mosaics

output_file_prefix
[str, optional] 'Text string that will be used as the filename prefix all files created by hapmultisequencer.py during the MVM custom mosaic generation process. If

not explicitly specified, all output files will start with the following formatted text string: "hst-skycell-p<pppp>-ra<##>d<####>-dec<n|s><##>d<####>", where p<pppp> is the projection cell ID, ra<##>d<####> are the whole-number and decimal portions of the right ascention, respectively, and dec<n|s><##>d<####> are the whole-number and decimal portions of the declination, respectively. Note that the "<n|s>" denotes if the declination is north (positive) or south (negative). Example: For skycell = 1974, ra = 201.9512, and dec = +26.0012, The filename prefix would be "skycell-p1974-ra201d9512-decn26d0012".

log_level
[int, optional] The desired level of verboseness in the log statements displayed on the screen and written to the .log file. Default value is 20, or 'info'.

Returns

return_value: integer
A return exit code used by the calling Condor/OWL workflow code: 0 (zero) for success, 1 for error

haputils.cell_utils

Module for defining ProjectionCells and SkyCells for a set of exposures.

`drizzlepac.haputils.cell_utils.`**`get_sky_cells`**(*visit_input*, *input_path=None*, *scale=None*, *cell_size=None*, *diagnostic_mode=False*)

Return all sky cells that overlap the exposures in the input.

Parameters

visit_input
[str or list] Input specifying the exposures from a single visit; either a poller output file or a simple list of exposure filenames. Exposures in an input list are assumed to be in the current working directory when running the code, unless `input_path` has been provided which points to the location of the exposures to be processed.

input_path
[str, optional] Location of input exposures, if provided. If not provided, location will be assumed to be the current working directory.

scale
[float, optional] User-defined value for the pixel scale, in arcseconds/pixel, of the sky cells and projection cells. If `None`, default value from grid definition file will be used.

cell_size
[float, optional] User-specified size, in degrees, for each projection cell. If `None`, default value from grid definition file will be used.

diagnostic_mode
[bool, optional] Parameter which specifies whether or not to generate additional output useful for debugging internal computations.

Returns

sky_cells
[list of objects] List of *SkyCell* objects for all sky cells which overlap the exposures provided in `visit_input`.

drizzlepac.haputils.cell_utils.**interpret_scells**(*sky_cells*)

Return dict of filenames each with the skycell name they overlap

Parameters

sky_cells
[dict] Dictionary of sky-cell objects from *get_sky_cells*

Returns

sky_cell_files
[dict] Dictionary of ALL sky-cell IDs as a ';'-delimited string for each exposure(sky cell member), with exposure filenames as keys.

class drizzlepac.haputils.cell_utils.**ProjectionCell**(*index=None*, *band=None*, *scale=None*, *nxy=None*, *overlap=None*)

Class which defines a projection cell product based on the position on the sky using the PanSTARRs-derived definitions.

This class has the following members for defining the projection cell:

- **wcs** : astropy.wcs.WCS object defining the WCS for the cell
- **footprint** : footprint of the projection cell on the sky from **wcs.calc_footprint()**
- **corners** : (RA, DEC) of the corners of the projection cell on the sky.

Build projection cell for cell with name `skycell_NNNNN`

If `band` is not specified, it will open the grid definitions file to obtain the band definition for the cell with the specified `index`.

Parameters

index
[int] ProjectionCell index on the sky. This index is 0-based. It either represents the index in a single declination band of ProjectionCells or the ID number of the ProjectionCell across the entire sky (from 0 to 2643, given the default GridDefs table).

band
[list, optional] Definition of spacing of projection cells along a line of constant declination. These defintions, if not provided, are extracted from the projection cell FITS table.

scale
[float, optional] Plate scale to use for defining the projection cell WCS. If not specified here, will use values from the projection cell FITS table *'PC_SCALE'* header keyword.

nxy
[int, optional] Number of sky cells in X and Y to use in subdividing the projection cell. If not specified here, will use values from the projection cell FITS table *'SC_NXY'* header keyword.

overlap
[int, optional] Number of pixels of overlap between sky cells. If not specified here, will use values from the projection cell FITS table *'SC_OLAP'* header keyword.

class drizzlepac.haputils.cell_utils.**SkyCell**(*projection_cell=None*, *x=None*, *y=None*, *scale='fine'*)

Definition of the sky cell within a ProjectionCell object.

This class defines the following attributes of a sky cell:

- **sky_cell_id** : string representation of the sky cell ID
- **projection_cell** : name of projection cell this sky cell belongs to
- **wcs**
  [astropy.wcs.WCS definition of the WCS of this sky cell as a] subarray within the tangent plane defined by the projection cell's WCS
- **corners** : (RA, Dec) array of the edges of the exposures within the sky cell
- **mask** : polygon representation exposures within the sky cell

Define sky cell at position x,y within projection cell.

Parameters

projection_cell
[object, optional] ProjectionCell instance which this SkyCell will be based upon.

x
[int, optional] X position (1-based) of this SkyCell within the ProjectionCell

y
[int, optional] Y position (1-based) of this SkyCell within the ProjectionCell

scale
[str or float, optional] Plate scale to be used to define the SkyCell WCS. The strings 'fine' or 'coarse' can be used to refer to default plate scales. Default values are specified in the dict `cell_utils.SUPPORTED_SCALES`. Alternatively, floating-point values can be provided to specify the exact pixel size in arcseconds/pixel should be used.

Examples

The SkyCell object can be initialized in one of 2 ways.

1. Using an already existing SkyCell name with:

```
>>> skycell = SkyCell.from_name('skycell-p0197x18y16')
```

2. Using separate projection cell ID, along with X and Y indices with:

```
>>> skycell = SkyCell(projection_cell=ProjectionCell(index=197), x=18, y=16)
```

haputils.make_poller_files

Module for creating CSV-formatted input 'poller' files for MVM processing. Generates a poller file that will be used as input to runsinglehap.py, hapsequencer.py, runmultihap.py or hapmultisequencer.py based on the files or rootnames listed user-specified list file.

USAGE

```
>>> python drizzlepac/haputils/make_poller_files.py <input filename> -[ost]
- input filename: Name of a file containing a list of calibrated fits files␣
↪(ending with "_flt.fits" or
  "_flc.fits") or rootnames (9 characters, usually ending with a "q" to␣
↪process. The corresponding
  flc.fits or flt.fits files must exist in the user-specified path, the␣
↪current working directory or the
  online cache
```

- The '-o' optional input allows users to input the name of an output poller file that will be created. If not explicitly specified, the poller file will be named "poller_file.out".
- The '-s' optional input allows users to input the Name of the skycell. The correct syntax for skycell names is "skycell-pNNNNxXXyXX", where NNNN is the 4-digit projection cell number, and XX and YY are the two-digit X and Y skycell indices, respectively. NOTE: this input argument is not needed for SVM poller file creation, but *REQUIRED* for MVM poller file creation. Users can determine the skycell(s) that their observations occupy using the `haputils.which_skycell` script.
- The '-t' optional input allows users to specify the type of poller file that will be created. The valid input options are "svm" to create a poller file for use with the single-visit mosaics pipeline or "mvm" to create a poller file for use with the multiple-visit mosaics pipeline. If not explicitly specified, the default value is "svm". NOTE: if creating a MVM poller file, one must specify the skycell name using the "-s" input argument.

Python USAGE:

```
>>> python
>>> from drizzlepac.haputils import make_poller_files
>>> make_poller_files.generate_poller_file(input_list, poller_file_type='svm
↪', output_poller_filename="poller_file.out", skycell_name=None):
```

drizzlepac.haputils.make_poller_files.**generate_poller_file**(*input_list*, *poller_file_type='svm'*, *output_poller_filename='poller_file.out'*, *skycell_name=None*)

Creates a properly formatted SVM or MVM poller file.

Parameters

input_list
[str] Name of the text file containing the list of filenames or rootnames to process

poller_file_type
[str, optional] Type of poller file to create. 'svm' for single visit mosaic, 'mvm' for multi-visit mosaic. Default value is 'svm'.

output_poller_filename
[str, optional] Name of the output poller file that will be created. Default value is 'poller_file.out'.

skycell_name
[str, optional] Name of the skycell to use when creating a MVM poller file. skycell_name is REQUIRED for the creation of a MVM poller file, but completely unnecessary for the creation of a SVM poller file. The correct syntax for skycell names is 'skycell-pNNNNxXXyXX', where NNNN is the 4-digit projection cell number, and XX and YY are the two-digit X and Y skycell indices, respectively. Default value is logical 'None'. NOTE: this input argument is not needed for SVM poller file creation, but *REQUIRED* for MVM poller file creation. Users can determine the skycell(s) that their observations occupy using the `haputils.which_skycell` script.

Returns

Nothing.

haputils.poller_utils

Module for interpreting input 'poller' files for MVM processing.

drizzlepac.haputils.poller_utils.**interpret_mvm_input**(*results*, *log_level*, *layer_method='all'*, *exp_limit=2.0*, *user_table=None*, *include_small=True*, *only_cte=False*)

Parameters

results
: [str or list] Input poller file name or Python list of dataset names to be processed for a multi-visit. Dataset names have to be either the filename of a singleton (non-associated exposure) or the name of an ASN file (e.g., jabc01010_asn.fits).

log_level
: [int] The desired level of verboseness in the log statements displayed on the screen and written to the .log file.

exp_limit
: [float, optional] This specifies the ratio between long and short exposures that should be used to define each exposure time bin (short, med, long). If None, all exposure times will be combined into a single bin (all). Splitting of the bins gets computed using Kmeans clustering for 3 output bins.

layer_method
: [str] Specify what type of interpretation should be done for the definition of each layer. Options include: 'all', 'hard', 'kmeans'. The value 'all' simply puts all exposures regardless of exposure time. The value of 'hard' relies on pre-defined limits for 'short', 'med', and 'long' exposure times. The value of 'kmeans' relies on using 'kmeans' analysis for defining sub-layers based on exposure time.

user_table
: [dict, optional] User-supplied table generated using *build_poller_table*. This table may have been edited to meet the unique needs of the user for this custom processing run.

include_small
: [bool, optional] Specify whether or not to create products from ACS/HRC and ACS/SBC data.

only_cte
: [bool, optional] Specify whether or not to create products only using CTE-corrected data.

Notes

Interpret the contents of the MVM poller file for the values to use in generating the names of all the expected output products.

Input will have format of:

```
ib4606c5q_flc.fits,11665,B46,06,1.0,F555W,UVIS,skycell-p2381x05y09,NEW,/ifs/
↪archive/ops/hst/public/ib46/ib4606c5q/ib4606c5q_flc.fits
```

which are:

```
filename, proposal_id, program_id, obset_id, exptime, filters, detector,␣
↪skycell-p<PPPP>x<XX>y<YY>, [OLD|NEW], pathname
```

The SkyCell ID will be included in this input information to allow for grouping of exposures into the same SkyCell layer based on filter, exptime, and year.

Output dict will have only have definitions for each defined sky cell layer to be either created or updated based on the input exposures being processed. There will be a layer defined for each unique combination of filter/exp class/year. An example would be:

```
obs_info_dict["layer 00": {"info": '11665 06 wfc3 uvis f555w short 2011 drc
↪',
                           "files": ['ib4606c5q_flc.fits', 'ib4606cgq_flc.
↪fits']}

obs_info_dict["layer 01": {"info": '11665 06 wfc3 uvis f555w med 2011 drc',
                           "files": ['ib4606c6q_flc.fits', 'ib4606cdq_flc.
↪fits']}

obs_info_dict["layer 02": {"info": '11665 06 wfc3 uvis f555w long 2011 drc',
                           "files": ['ib4606c9q_flc.fits', 'ib4606ceq_flc.
↪fits']}
```

Caution: The input ("results") parameter was originally designed to be able to accept a Python list of pipeline product dataset names (e.g., ib4606cdq_flc.fits) for MVM proessing. However, this was later modified. MVM processing needs the SVM FLT/FLC files as input.

haputils.product

Module for defining classes for MVM processing.

drizzlepac.haputils.product.**SkyCellExposure**(*prop_id*, *obset_id*, *instrument*, *detector*, *aperture_from_poller*, *filename*, *layer*, *filetype*, *log_level*)

An SkyCell Exposure Product is an individual exposure/image (flt/flc).

The "sce" is short hand for ExposureProduct.

drizzlepac.haputils.product.**SkyCellProduct**(*prop_id*, *obset_id*, *instrument*, *detector*, *aperture_from_poller*, *skycell_name*, *layer*, *filetype*, *log_level*)

A SkyCell Product is a mosaic comprised of images acquired during a multiple visits with one instrument, one detector, a single filter, and all exposure times. The exposure time characteristic may change - TBD.

The "scp" is short hand for SkyCellProduct.

haputils.generate_custom_svm_mvm_param_file

Module for defining custom sets of parameters for SVM and MVM processing. generate_custom_mvm_svm_param_file - a module to create a template SVM/MVM processing pipeline parameter file based on the observations present in the current working directory for the user to customize

Command-line USAGE:

```
>>> python generate_custom_svm_mvm_param_file.py [-clop] poller_filename
```

- poller_filename: (required) the name of the input SVM/MVM poller file
- -c: (optional) If turned on, existing files with the same name as the output custom SVM parameter file created by this script will be overwritten.
- -l: (optional) The desired level of verboseness in the log statements displayed on the screen and written to the .log file. Valid input options are 'critical', 'error', 'warning', 'info', or 'debug'.
- -o: (optional) Name of the output configuration JSON file which will be created for specialized processing. This file will contain ALL the input parameters necessary for processing. If not explicitly specified, the default name for the output file is "custom_parameters.json".
- -p: (optional) Name of the pipeline that the configurations will be prepared for. Valid options are 'mvm' (for the HAP multi-visit mosaics pipeline) or 'svm' (for the HAP single-visit mosaic pipeline). If not explicitly stated, the default value is 'svm'

Python USAGE:

```
>>> python
>>> from drizzlepac.haputils import generate_custom_svm_mvm_param_file
>>> generate_custom_svm_mvm_param_file.make_svm_input_file(input_filename,
                                                          clobber=False,
                                                          log_level=logutil.
↪logging.INFO
                                                          output_custom_pars_
↪file='custom_parameters.json',
                                                          hap_pipeline_name=
↪'svm')
```

Note: This script only generates a template input parameter file populated with default values based on the observations present in the current working directory. It is entirely incumbent on the user to modify this file with non-default values.

drizzlepac.haputils.generate_custom_svm_mvm_param_file.**make_svm_input_file**(*input_filename*, *hap_pipeline_name='svm*
output_custom_pars_file='cu
clobber=False,
log_level=20)

create a custom SVM processing pipeline parameter file based on the observations present in the current working directory using config_utils.HapConfig() and optionally update_ci_values() to adjust CI upper and lower limits for filter products

Parameters

input_filename: str
The 'poller file' where each line contains information regarding an exposures taken during a single visit.

hap_pipeline_name
[str, optional] Name of the pipeline that the configurations will be prepared for. Valid options are 'mvm' (for the HAP multi-visit mosaics pipeline) or 'svm' (for the HAP single-visit mosaic pipeline). If not explicitly stated, the default value is 'svm'

output_custom_pars_file: str, optional
Fully specified output filename which contains all the configuration parameters available during the processing session. Default is 'custom_parameters.json'.

clobber
[Bool, optional] If set to Boolean 'True', existing files with the same name as *output_custom_pars_file*, the output custom SVM parameter file created by this script will be overwritten. Default value is Boolean 'False'.

log_level
[int, optional] The desired level of verboseness in the log statements displayed on the screen and written to the .log file. Default value is 20, or 'info'.

Returns

Nothing.

The MVM processing can be performed using one of two interfaces:

- command-line interface: *runmultihap*
- Python interface: *hapmultisequencer*

Command-line Interface for MVM Processing

The task `runmultihap` serves as the primary interface for processing data from multiple visits that share a single skycell into a uniform set of images. runmultihap.py - Module to control processing of multi-visit mosaics

License
License

USAGE:

```
>>> python drizzlepac/runmultihap [-dl] inputFilename
```

- The '-d' option will run this task in DEBUG mode producing additional outputs. If not explicitly specified, the default value is logical "False".
- The '-l' option allows the user to control the level of verboseness in the log statements displayed on the screen and written to the .log file. Valid options are "critical", "error", "warning", "info", and "debug". If not explicitly specified, the default value is "info".

Python USAGE:

```
>>> python
>>> from drizzlepac import runmultihap
>>> runmultihap.perform(inputFilename, debug=False, log_level="info")
```

drizzlepac.runmultihap.**perform**(*input_filename*, ***kwargs*)

Main calling subroutine.

Parameters

input_filename
[string] Name of the input csv file containing information about the files to be processed

debug
[Boolean] display all tracebacks, and debug information?

input_custom_pars_file
[str, optional] Represents a fully specified input filename of a configuration JSON file which has been customized for specialized processing. This file should contain ALL the input parameters necessary for processing. If not specified, default configuration parameter values will be used.

log_level
[string] The desired level of verboseness in the log statements displayed on the screen and written to the .log file.

Associated Helper Code

These modules and functions assist with some of the logistics associated with multi-visit processing.

drizzlepac.haputils.which_skycell

Reports the name(s) of the projection cell(s)/skycell(s) that the observations in the user-specified input file occupy.

USAGE:

```
>>> python drizzlepac/haputils/which_skycell.py -[i]
```

- The '-i' optional input allows users to specify the name of a file containing a list of calibrated fits files (ending with "_flt.fits" or "_flc.fits") to process. If not explicitly specified, all 'flc/flt fits files in the current working directory will be processed.

Python USAGE:

```
>>> python
>>> from drizzlepac.haputils import which_skycell
>>> which_skycell.report_skycells(img_list)
```

drizzlepac.haputils.which_skycell.**report_skycells**(*img_list*)

reports the names of the projection cell(s)/skycell(s) that the observations in the user-specified input list occupy.

Parameters

img_list
[list] list of the images to process

Returns

Nothing!

API for MVM Utilities

The `hapcut_utils.py` module is part of the drizzlepac package and contains utilities which are intended to be used with Multi-visit Mosaic (MVM) data and to be imported into other Python programs or a Python session for interactive data analysis.

haputils.hapcut_utils

The module is a high-level interface to astrocut for use with HAP SVM and MVM files.

drizzlepac.haputils.hapcut_utils.**mvm_id_filenames**(*sky_coord*, *cutout_size*, *log_level=20*)

This function retrieves a table of MVM drizzled image filenames with additional information from the archive. The user can then further cull the table to use as input to obtain a list of files from the archive. This function will return filter-level products. At this time, both ACS and WFC3 are searched by default.

Parameters

sky_coord
[str or astropy.coordinates.SkyCoord object] The position around which to cutout. It may be specified as a string ("ra dec" in degrees) or as the appropriate astropy.coordinates.SkyCoord object.

cutout_size
[int, array-like, astropy.units.Quantity] The size of the cutout array. If `cutout_size` is a scalar number or a scalar astropy.units.Quantity, then a square cutout of `cutout_size` will be created. If `cutout_size` has two elements, they

should be in `(ny, nx)` order. Scalar numbers in `cutout_size` are assumed to be in units of arcseconds. astropy.units.Quantity objects must be in angular units.

log_level
[int, optional] The desired level of verbosity in the log statements displayed on the screen and written to the .log file. Default value is 20, or 'info'.

Returns

final_table
[astropy.table.Table object]

**This utility also writes an output ECSV file version of the in-memory filtered data product table,
final_table. The output filename is in the form:
mvm_query-ra<###>d<####>-dec<n|s><##>d<####>_<radius>_cutout.ecsv
(e.g., mvm_query-ra84d8208-decs69d8516_354_cutout.ecsv, where radius has been computed from the
cutout dimensions.**

drizzlepac.haputils.hapcut_utils.**mvm_retrieve_files**(*products*, *archive=False*, *clobber=False*, *log_level=20*)

This function retrieves specified files from the archive - unless the file is found to be locally resident on disk. Upon completion, The function returns a list of filenames available on disk.

Parameters

products
[astropy.table.Table object] A Table of products as returned by the mvm_id_filenames function.

archive
[Boolean, optional] Retain copies of the downloaded files in the astroquery created sub-directories? Default is "False".

clobber
[Boolean, optional] Download and Overwrite existing files? Default is "False".

log_level
[int, optional] The desired level of verbosity in the log statements displayed on the screen and written to the .log file. Default value is 20, or 'info'.

Returns

local_files
[list] List of filenames

Note: Code here cribbed from retrieve_obsevation in astroquery_utils module.

drizzlepac.haputils.hapcut_utils.**make_the_cut**(*input_files*, *sky_coord*, *cutout_size*, *output_dir='.'*, *log_level=20*, *verbose=False*)

This function makes the actual cut in the input MVM drizzled filter- and exposure-level FITS files. As such it is a high-level interface for the astrocut.cutouts.fits_cut functionality.

Parameters

input_files
[list] List of fits image filenames from which to create cutouts.

sky_coord
[str or astropy.coordinates.SkyCoord object] The position around which to cutout. It may be specified as a string ("ra dec" in degrees) or as the appropriate astropy.coordinates.SkyCoord object.

cutout_size
[int, array-like, astropy.units.Quantity] The size of the cutout array. If `cutout_size` is a scalar number or a scalar astropy.units.Quantity, then a square cutout of `cutout_size` will be created. If `cutout_size` has two elements, they should be in `(ny, nx)` order. Scalar numbers in `cutout_size` are assumed to be in units of arcseconds. astropy.units.Quantity objects must be in angular units.

output_dir
[str] Default value '.'. The directory where the cutout file(s) will be saved.

log_level
[int, optional] The desired level of verbosity in the log statements displayed on the screen and written to the .log file. Default value is 20, or 'info'.

verbose
[bool] Default False. If True, additional intermediate information is printed for the underlying spacetelescope.astrocut utilities.

Returns

response
[list] Returns a list of all the output filenames.

Note: For each input file designated for a cutout, there will be a corresponding output file.
Since both the SCI and WHT extensions of the input files are actually cut, individual fits files will contain two image extensions, a SCI followed by the WHT.

While the standard pipeline processing does not produce an MVM exposure-level drizzled product, it is possible for a user to turn on this capability in the pipeline while performing custom processing. As such this routine will perform cutouts of the exposure-level drizzled files.

Each filter-level output filename will be of the form:
hst_cutout_skycell-p<pppp>-ra<##>d<####>-dec<n|s><##>d<####>_instrument_detector_filter[_platescale].fits

Each exposure-level filename will be of the form:
hst_cutout_skycell-p<pppp>-ra<##>d<####>-dec<n|s><##>d<####>_instrument_detector_filter[_platescale]-ipppssoo.fits

where platescale has the value of "coarse" representing 0.12"/pixel for WFC3/IR, or there is no platescale value present which is the default and represents a "fine" platescale of 0.04"/pixel.

drizzlepac.haputils.hapcut_utils.**mvm_combine**(*cutout_files*, *output_dir='.'*, *log_level=20*)

This function combines multiple MVM skycell cutout images from the same detector/filter combination to create a single view of the requested data. All of the functions in this module are designed to work in conjunction with one another, so the cutout images should be on the user's local disk. This task is a high-level wrapper for the astrocut.cutout_processing.combine functionality.

Specifically, this routine will combine filter-level cutouts from multiple skycells, all sharing the same detector and filter. This routine will also combine exposure-level cutouts from multiple skycells, all sharing the same detector, filter, and ipppssoo. Images which do not share a detector and filter with any other image will be ignored. Individual exposures from a single skycell will also be ignored.

Parameters

cutout_files
[list] List of fits image cutout filenames where the cutouts are presumed to have been created with *make_the_cut*.

output_dir
[str] Default value '.' - The directory where the output combined files will be saved.

log_level
[int, optional] The desired level of verbosity in the log statements displayed on the screen and written to the .log file. Default value is 20, or 'info'.

API for make_custom_mosaic

The task `make_custom_mosaic` serves as the primary interface for processing data from a multiple visits that span multiple skycells into a uniform set of images.

Frequently Asked Questions

- **Why would a user want to run this code?**
  - Standard multi-visit mosaic (MVM) processing only processes a single skycell per run. Datasets with images that span one or more skycell boundaries require multiple MVM processing runs and produce multiple overlapping product images that are segmented at skycell boundaries. This code simplifies this process. It creates seamless multi-visit mosaics for input datasets that span one or more skycell boundaries.
- **What are the required and opitonal inputs? What do they do?**
  - The only required to make_custom_mosaic.py is either the name of a text file containing a list of calibrated ACS or WFC _flc.fits or _flt.fits files process or a search search sting that will be used to find input files. Optional input arguments control the level of verbosity in the log statements that are displayed to the screen and written to log files, the formatting of output filenames, and the method used to align input images. These inputs are discussed in more detail in the **USAGE** section.
- **When the code is run, what output products should be expected? Do they differ depending on the inputs?**

- As this script depends on `hapmultisequencer` to perform the actual image processing, users can expect the output products of make_custom_mosaics.py to be functionally identical to standard MVM products. As with standard MVM processing, the only variation in output images occurs when processing WFC3/IR data, which produces output imagery products at the standard platescale of 0.04 arcseconds per pixel, as well as at the native WFC3/IR detector platescale of 0.12 arcseconds per pixel.

make_custom_mosaic.py - Module to control the generation of user-defined multi-skycell mosaics. This script extends the capabilities of the HAP multi-visit mosaics (MVM) processing code (hapmultisequencer.py). Based on user input, this script will produce drizzle-combined mosaics that will be as large as necessary to incorporate all input images. Mosaic images can span multiple skycells, but NOT multiple projection cells. In the exceedingly rare case that a custom mosaic contains observations that fall on multiple adjacent projection cells, the bounds of the mosaic will be automatically clipped at the projection cell edge.

As this script is simply a wrapper around hapmultisequencer.py, the final output products are the exact same as what one would expect from a pipeline MVM run. For a complete details about all hapmultisequencer.py inputs and output products, please refer to the hapmultisequencer.py documentation.

hapmultisequencer.py was explicitly written to process data from only a single skycell. However, this script gives users the ability to create mosaics spanning multiple skycells. As such, all output products produced by hapmultisequencer.py normally include both the projection cell and skycell names in the all output products. To avoid any confusion that might arise during the creation of mosaics that span multiple skycells, all custom mosaic output product filenames will by default include projection cell name and the projection cell reference right ascension and declination instead of the skycell name. Users can also optionally specify an output product filename prefix of their choosing.

The output world cooordinate system (WCS) information is based on that of the projection cell in which the observations reside. If the input observations happen to fall in a region of the sky where more than one projection cells overlap, the WCS information of the output products will be based on the projection cell whose center is closest to the geometric center of the input observations.

For all detectors, output products will be generated with a plate scale of 0.04 arcseconds per pixel. The script produces an additional set up output products for WFC3/IR observations. This additional set of products is generated using the native detector platescale of 0.12 arcseconds per pixel. These files end in "_coarse_all_flt.fits".

The output world coordinate system (WCS) information is based on that of the projection cell in which the observations reside. If the input observations happen to fall in a region of the sky where more than one projection cells overlap, the WCS information of the output products will be based on the projection cell whose center is closest to the geometric center of the input observations.

USAGE:

```
>>> python drizzlepac/make_custom_mosaic.py -[los] <search pattern enclosed␣
↪in quotes>
```

or

```
>>> python drizzlepac/make_custom_mosaic.py -[los] <text file with list of␣
↪input files>
```

- The '-l' option allows the user to control the level of verboseness in the log statements displayed on the screen and written to the .log file. The level of verboseness from left to right, and includes all log statements with a log_level left of the specified level. Specifying "critical" will only record/display "critical" log statements, and specifying "error" will record/display both "error" and "critical" log statements, and so on.
- The '-o' option allows the user to specify the text string that will be used as the filename prefix all files created by hapmultisequencer.py during the MVM custom mosaic generation process. If not explicitly specified, all output files will start with the following formatted text string: "hst-skycell-p<pppp>-ra<##>d<####>-dec<n|s><##>d<####>", where p<pppp> is the projection cell ID, ra<##>d<####> are the whole-number and decimal portions of the right ascention, respectively, and dec<n|s><##>d<####> are the whole-number and decimal portions of the declination, respectively. Note that the "<n|s>" denotes if the declination is north (positive) or south (negative). Example: For skycell = 1974, ra = 201.9512, and dec = +26.0012, The filename prefix would be "skycell-p1974-ra201d9512-decn26d0012".
- When turned on, the '-s' option allows users to skip alignment of all input images to known Gaia/HSC sources in the input image footprint and instead use the existing input image alignment solution.

Python USAGE:

```
>>> python
>>> from drizzlepac import make_custom_mosaic
>>> make_custom_mosaic.perform(<list file or search pattern>, log_level=
↪'info', output_file_prefix=None, skip_gaia_alignment=False)
```

drizzlepac.make_custom_mosaic.**calc_skycell_dist**(*x*, *y*, *x_ref*, *y_ref*)

Calculate distance from one skyframe to another

Parameters

x
[int] skyframe x index

y
[int] skyframe y index

x_ref
[int] reference x index

y_ref
[int] reference y index

Returns

dist
[float] distance from (x, y) to (x_ref, y_ref)

drizzlepac.make_custom_mosaic.**compute_mosaic_limits**(*proj_cell_dict*)

Compute min and max limits of the rectangle that encloses the mosaic observations

Parameters

proj_cell_dict
[dictionary] Dictionary containing projection cell information

Returns

mosaic_limits
[list] 4-element list containing the mosaic bounding rectangle X min and max and Y min and max values

drizzlepac.make_custom_mosaic.**create_input_image_list**(*user_input*)

Create list of input images based in user input from command-line

Parameters

user_input
[str] Search pattern to be used to identify images to process or the name of a text file containing a list of images to process

Returns

img_list
[list] list of images to process

drizzlepac.make_custom_mosaic.**create_poller_file**(*img_list*, *proj_cell_dict*)

Subroutine that executes make_poller_files.generate_poller_file() to generate custom MVM poller file for execution of hapmultisequencer.run_mvm_processing().

Parameters

img_list
[list] list of images to process

proj_cell_dict
[dict] Dictionary containing projection cell information

Returns

poller_filename
[str] Name of the newly created poller_filename

drizzlepac.make_custom_mosaic.**determine_projection_cell**(*img_list*)

Determine which projection cell should be used as the basis for the WCS of the output mosaic product(s) based on which projection cell center is closest to the center of the observations.

Parameters

img_list
[list] A list of images to process

Returns

best_pc_dict
[dict] Dictionary of SkyCell objects for the input observations. These SkyCell objects contain projection cell information from the projection cell whose center is closest to the geometric center of the input observations.

drizzlepac.make_custom_mosaic.**main**()

Command-line interface

Parameters

None

Returns

return_value
[int] return value from the run. 0 for successful run, something else otherwise.

drizzlepac.make_custom_mosaic.**perform**(*input_image_source*, *log_level='info'*, *output_file_prefix=None*, *skip_gaia_alignment=False*)

Main calling subroutine

Parameters

input_image_source
[str] Search pattern to be used to identify images to process or the name of a text file containing a list of images to process.

log_level
[str, optional] The desired level of verboseness in the log statements displayed on the screen and written to the .log file. The level of verboseness from left to right, and includes all log statements with a log_level left of the specified level. Specifying "critical" will only record/display "critical" log statements, and specifying "error" will record/display both "error" and "critical" log statements, and so on. Unless explicitly set, the default value is 'info'.

output_file_prefix
[str, optional] 'Text string that will be used as the filename prefix all files created by hapmultisequencer.py during the MVM custom mosaic generation process. If not explicitly specified, all output files will start with the following formatted text string: "hst-skycell-p<pppp>-ra<##>d<####>-dec<n|s><##>d<####>", where p<pppp> is the projection cell ID, ra<##>d<####> are the whole-number and decimal portions of the right ascention, respectively, and dec<n|s><##>d<####> are the whole-number and decimal portions of the declination, respectively. Note that the "<n|s>" denotes if the declination is north (positive) or south (negative). Example: For skycell = 1974, ra = 201.9512, and dec = +26.0012, The filename prefix would be "skycell-p1974-ra201d9512-decn26d0012".

skip_gaia_alignment
[bool, optional] Skip alignment of all input images to known Gaia/HSC sources in the input image footprint? If set to 'True', The existing input image alignment solution will be used instead. The default is False.

Returns

return_value
[int] A simple status value. '0' for a successful run or '1' for a failed run.

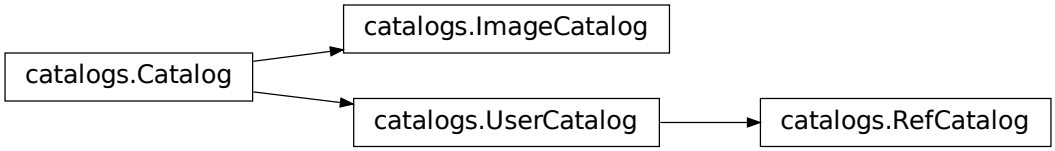
Classes

Classes to manage Catalogs and WCS's

This module provides the classes used to generate and manage source catalogs for each input chip. Those positions can be transformed to undistorted sky positions, written out to files, or plotted using various methods defined for these classes.

Authors
Warren Hack

License
License



drizzlepac.catalogs.**generateCatalog**(*wcs*, *mode='automatic'*, *catalog=None*, *src_find_filters=None*, ***kwargs*)

Function which determines what type of catalog object needs to be instantiated based on what type of source selection algorithm the user specified.

Parameters

wcs
[obj] WCS object generated by STWCS or PyWCS

catalog
[str or ndarray] Filename of existing catalog or ndarray of image for generation of source catalog.

kwargs
[dict] Parameters needed to interpret source catalog from input catalog with `findmode` being required.

Returns

catalog
[obj] A Catalog-based class instance for keeping track of WCS and associated source catalog

class drizzlepac.catalogs.**ImageCatalog**(*wcs*, *catalog_source*, *src_find_filters=None*, ***kwargs*)

Bases: *Catalog*

Class which generates a source catalog from an image using Python-based, daofind-like algorithms

Required input `kwargs` parameters:

```
computesig, skysigma, threshold, peakmin, peakmax,
hmin, conv_width, [roundlim, sharplim]
```

This class requires the input of a WCS and a source for the catalog, along with any arguments necessary for interpreting the catalog.

Parameters

wcs
[obj] Input WCS object generated using STWCS or HSTWCS

catalog_source
[str] Name of the file from which to read the catalog.

kwargs
[dict] Parameters for interpreting the catalog file or for performing the source extraction from the image. These will be set differently depending on the type of catalog being instantiated.

generateXY(***kwargs*)

Generate source catalog from input image using DAOFIND-style algorithm

class drizzlepac.catalogs.**UserCatalog**(*wcs*, *catalog_source*, ***kwargs*)

Bases: *Catalog*

Class to manage user-supplied catalogs as inputs.

Required input `kwargs` parameters:

```
xyunits, xcol, ycol[, fluxcol, [idcol]]
```

This class requires the input of a WCS and a source for the catalog, along with any arguments necessary for interpreting the catalog.

Parameters

wcs
[obj] Input WCS object generated using STWCS or HSTWCS

catalog_source
[str] Name of the file from which to read the catalog.

kwargs
[dict] Parameters for interpreting the catalog file or for performing the source extraction from the image. These will be set differently depending on the type of catalog being instantiated.

COLNAMES = ['xcol', 'ycol', 'fluxcol']

IN_UNITS = None

generateXY(***kwargs*)

Method to interpret input catalog file as columns of positions and fluxes.

plotXYCatalog(***kwargs*)

Plots the source catalog positions using matplotlib's `pyplot.plot()`

Plotting `kwargs` that can also be passed include any keywords understood by matplotlib's `pyplot.plot()` function such as:

```
vmin, vmax, cmap, marker
```

set_colnames()

Method to define how to interpret a catalog file Only needed when provided a source catalog as input

class drizzlepac.catalogs.**RefCatalog**(*wcs*, *catalog_source*, ***kwargs*)

Bases: *UserCatalog*

Class which manages a reference catalog.

Notes

A *reference catalog* is defined as a catalog of undistorted source positions given in RA/Dec which would be used as the master list for subsequent matching and fitting.

This class requires the input of a WCS and a source for the catalog, along with any arguments necessary for interpreting the catalog.

Parameters

wcs
[obj] Input WCS object generated using STWCS or HSTWCS

catalog_source
[str] Name of the file from which to read the catalog.

kwargs
[dict] Parameters for interpreting the catalog file or for performing the source extraction from the image. These will be set differently depending on the type of catalog being instantiated.

COLNAMES = ['refxcol', 'refycol', 'rfluxcol']

IN_UNITS = 'degrees'

PAR_NBRIGHT_PREFIX = 'ref'

PAR_PREFIX = 'r'

buildXY(*catalogs*)

generateRaDec()

Convert XY positions into sky coordinates using STWCS methods.

generateXY(***kwargs*)

Method to interpret input catalog file as columns of positions and fluxes.

class drizzlepac.catalogs.**Catalog**(*wcs*, *catalog_source*, ***kwargs*)

Bases: object

Base class for keeping track of a source catalog for an input WCS

> **Warning:** This class should never be instantiated by itself, as necessary methods are not defined yet.

This class requires the input of a WCS and a source for the catalog, along with any arguments necessary for interpreting the catalog.

Parameters

wcs
[obj] Input WCS object generated using STWCS or HSTWCS

catalog_source
[str] Name of the file from which to read the catalog.

kwargs
[dict] Parameters for interpreting the catalog file or for performing the source extraction from the image. These will be set differently depending on the type of catalog being instantiated.

PAR_NBRIGHT_PREFIX = ''

PAR_PREFIX = ''

apply_exclusions(*exclusions*)

Trim sky catalog to remove any sources within regions specified by exclusions file.

apply_flux_limits()

Apply any user-specified limits on source selection Limits based on fluxes.

buildCatalogs(*exclusions=None*, ***kwargs*)

Primary interface to build catalogs based on user inputs.

generateRaDec()

Convert XY positions into sky coordinates using STWCS methods.

generateXY(***kwargs*)

Method to generate source catalog in XY positions Implemented by each subclass

plotXYCatalog(***kwargs*)

Method which displays the original image and overlays the positions of the detected sources from this image's catalog.

Plotting `kwargs` that can be provided are:

vmin, vmax, cmap, marker

Default colormap is `summer`.

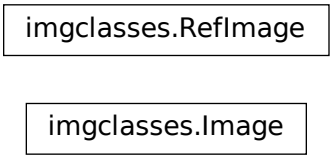
set_colnames()

Method to define how to interpret a catalog file Only needed when provided a source catalog as input

writeXYCatalog(*filename*)

Write out the X,Y catalog to a file

Image Class



Classes to keep track of all WCS and catalog information. Used by `TweakReg`.

Authors

Warren Hack, Mihai Cara

License

License

class drizzlepac.imgclasses.**Image**(*filename*, *input_catalogs=None*, *exclusions=None*, ***kwargs*)

Bases: `object`

Primary class to keep track of all WCS and catalog information for a single input image. This class also performs all matching and fitting.

Parameters

filename

[str] Filename for image.

input_catalogs

[list of str or None] Filename of catalog files for each chip, if specified by user.

kwargs
[dict] Parameters necessary for processing derived from input configObj object.

buildDefaultRefWCS()

Generate a default reference WCS for this image.

buildSkyCatalog()

Convert sky catalog for all chips into a single catalog for the entire field-of-view of this image.

clean()

Remove intermediate files created.

close()

Close any open file handles and flush updates to disk

compute_fit_rms()

get_shiftfile_row()

Return the information for a shiftfile for this image to provide compatability with the IRAF-based MultiDrizzle.

get_wcs()

Helper method to return a list of all the input WCS objects associated with this image.

get_xy_catnames()

Return a string with the names of input_xy catalog names

match(*refimage*, *quiet_identity*, ***kwargs*)

Uses xyxymatch to cross-match sources between this catalog and a reference catalog (refCatalog).

openFile(*openDQ=False*)

Open file and set up filehandle for image file

performFit(***kwargs*)

Perform a fit between the matched sources.

Parameters

kwargs
[dict] Parameter necessary to perform the fit; namely, *fitgeometry*.

Notes

This task still needs to implement (eventually) interactive iteration of
the fit to remove outliers.

sortSkyCatalog()

Sort and clip the source catalog based on the flux range specified by the user. It keeps a copy of the original full list in order to support iteration.

transformToRef(*ref_wcs*, *force=False*)

Transform sky coords from ALL chips into X,Y coords in reference WCS.

updateHeader(*wcsname='TWEAK'*, *reusename=False*)

Update header of image with shifts computed by *perform_fit()*.

writeHeaderlet(***kwargs*)

Write and/or attach a headerlet based on update to PRIMARY WCS

write_fit_catalog()

Write out the catalog of all sources and resids used in the final fit.

write_outxy(*filename*)

Write out the output(transformed) XY catalog for this image to a file.

write_skycatalog(*filename*)

Write out the all_radec catalog for this image to a file.

class drizzlepac.imgclasses.**RefImage**(*wcs_list*, *catalog*, *xycatalog=None*, *cat_origin=None*, ***kwargs*)

Bases: object

This class provides all the information needed by to define a reference tangent plane and list of source positions on the sky.

Warning: When `wcs_list` is a Python list of `WCS` objects, each element must be an instance of `stwcs.wcsutil.HSTWCS`.

append_not_matched_sources(*image*)

clean()

Remove intermediate files created

clear_dirty_flag()

close()

get_shiftfile_row()

Return the information for a shiftfile for this image to provide compatability with the IRAF-based MultiDrizzle.

set_dirty()

transformToRef()

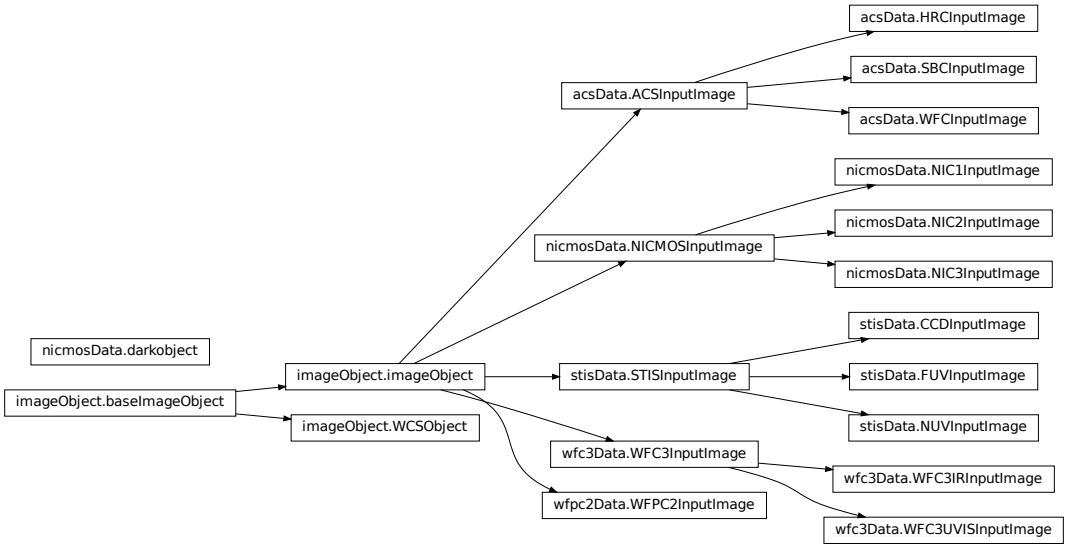
Transform reference catalog sky positions (self.all_radec) to reference tangent plane (self.wcs) to create output X,Y positions.

write_skycatalog(*filename*, *show_flux=False*, *show_id=False*)

Write out the all_radec catalog for this image to a file.

imageObject Classes

This class and related sub-classes manage all the instrument-specific images for processing by *astrodrizzle*.



Base ImageObject Classes

A class which makes image objects for each input filename. A class which makes image objects for each input filename.

Authors
Warren Hack

License
License

class drizzlepac.imageObject.**baseImageObject**(*filename*)

Bases: object

Base ImageObject which defines the primary set of methods.

buildERRmask(*chip*, *dqarr*, *scale*)

Builds a weight mask from an input DQ array and an ERR array associated with the input image.

buildEXPmask(*chip*, *dqarr*)

Builds a weight mask from an input DQ array and the exposure time per pixel for this chip.

buildIVMmask(*chip*, *dqarr*, *scale*)

Builds a weight mask from an input DQ array and either an IVM array provided by the user or a self-generated IVM array derived from the flat-field reference file associated with the input image.

buildMask(*chip*, *bits=0*, *write=False*)

Build masks as specified in the user parameters found in the configObj object.

We should overload this function in the instrument specific implementations so that we can add other stuff to the badpixel mask? Like vignetting areas and chip boundries in nicmos which are camera dependent? these are not defined in the DQ masks, but should be masked out to get the best results in multidrizzle.

clean()

Deletes intermediate products generated for this imageObject.

close()

Close the object nicely and release all the data arrays from memory YOU CANT GET IT BACK, the pointers and data are gone so use the getData method to get the data array returned for future use. You can use putData to reattach a new data array to the imageObject.

findExtNum(*extname=None*, *extver=1*)

Find the extension number of the give extname and extver.

find_DQ_extension()

Return the suffix for the data quality extension and the name of the file which that DQ extension should be read from.

getAllData(*extname=None*, *exclude=None*)

This function is meant to make it easier to attach ALL the data extensions of the image object so that we can write out copies of the original image nicer.

If no extname is given, the it retrieves all data from the original file and attaches it. Otherwise, give the name of the extensions you want and all of those will be restored.

Ok, I added another option. If you want to get all the data extensions EXCEPT a particular one, leave extname=NONE and set exclude=EXTNAME. This is helpfull cause you might not know all the extnames the image has, this will find out and exclude the one you do not want overwritten.

getData(*exten=None*)

Return just the data array from the specified extension fileutil is used instead of fits to account for non- FITS input images. openImage returns a fits object.

getExtensions(*extname='SCI'*, *section=None*)

Return the list of EXTVER values for extensions with name specified in extname.

getGain(*exten*)

getHeader(*exten=None*)

Return just the specified header extension fileutil is used instead of fits to account for non-FITS input images. openImage returns a fits object.

getInstrParameter(*value*, *header*, *keyword*)

This method gets a instrument parameter from a pair of task parameters: a value, and a header keyword.

The default behavior is:

- if the value and header keyword are given, raise an exception.
- if the value is given, use it.
- if the value is blank and the header keyword is given, use the header keyword.
- if both are blank, or if the header keyword is not found, return None.

getKeywordList(*kw*)

Return lists of all attribute values for all active chips in the `imageObject`.

getNumpyType(*irafType*)

Return the corresponding numpy data type.

getOutputName(*name*)

Return the name of the file or PyFITS object associated with that name, depending on the setting of self.inmemory.

getReadNoiseImage(*chip*)

Notes

Method for returning the readnoise image of a detector (in electrons).

The method will return an array of the same shape as the image.

Units
electrons

getdarkcurrent()

Notes

Return the dark current for the detector. This value will be contained within an instrument specific keyword. The value in the image header will be converted to units of electrons.

Units
electrons

getdarkimg(*chip*)

Notes

Return an array representing the dark image for the detector.

The method will return an array of the same shape as the image.

Units
electrons

`getexptimeimg`(*chip*)

Returns

exptimeimg
[numpy array] The method will return an array of the same shape as the image.

Notes

Return an array representing the exposure time per pixel for the detector. This method will be overloaded for IR detectors which have their own EXP arrays, namely, WFC3/IR and NICMOS images.

Units
None

`getflat`(*chip*, *flat_file=None*, *flat_ext=None*)

Method for retrieving a detector's flat field.

Parameters

chip
[int] Chip number. Same as FITS `EXTVER`.

flat_file
[str, None] Flat field file name. If not specified, it will be determined automatically from image header.

flat_ext
[str, None] Flat field extension name (same as FITS `EXTNAME`). Specifies extension name containing flat field data.

Returns

flat: array
This method will return an array the same shape as the image in **units of electrons**.

`getskyimg`(*chip*)

Notes

Return an array representing the sky image for the detector. The value of the sky is what would actually be subtracted from the exposure by the skysub step.

Units
electrons

info()

Return fits information on the _image.

putData(*data=None*, *exten=None*)

Now that we are removing the data from the object to save memory, we need something that cleanly puts the data array back into the object so that we can write out everything together using something like fits.writeto….this method is an attempt to make sure that when you add an array back to the .data section of the hdu it still matches the header information for that section (ie. update the bitpix to reflect the datatype of the array you are adding). The other header stuff is up to you to verify.

Data should be the data array exten is where you want to stick it, either extension number or a string like 'sci,1'

returnAllChips(*extname=None*, *exclude=None*)

Returns a list containing all the chips which match the extname given minus those specified for exclusion (if any).

saveVirtualOutputs(*outdict*)

Assign in-memory versions of generated products for this `imageObject` based on dictionary 'outdict'.

set_mt_wcs(*image*)

Reset the WCS for this image based on the WCS information from another imageObject.

set_units()

Record the units for this image, both BUNITS from header and in_units as needed internally. This method will be defined specifically for each instrument.

set_wtscl(*chip*, *wtscl_par*)

Sets the value of the wt_scl parameter as needed for drizzling.

updateContextImage(*contextpar*)

Reset the name of the context image to `None` if parameter `context` is `False`.

updateData(*exten*, *data*)

Write out updated data and header to the original input file for this object.

updateIVMName(*ivmname*)

Update outputNames for image with user-supplied IVM filename.

updateOutputValues(*output_wcs*)

Copy info from output WCSObject into outputnames for each chip for use in creating outputimage object.

class drizzlepac.imageObject.**imageObject**(*filename*, *output=None*, *group=None*, *inmemory=False*)

Bases: *baseImageObject*

This returns an imageObject that contains all the necessary information to run the image file through any multidrizzle function. It is essentially a PyFits object with extra attributes.

There will be generic keywords which are good for the entire image file, and some that might pertain only to the specific chip.

compute_wcslin(*undistort=True*)

Compute the undistorted WCS based solely on the known distortion model information associated with the WCS.

setInstrumentParameters(*instrpars*)

Define instrument-specific parameters for use in the code. By definition, this definition will need to be overridden by methods defined in each instrument's sub-class.

set_units(*chip*)

Define units for this image.

class drizzlepac.imageObject.**WCSObject**(*filename*, *suffix='_drz'*)

Bases: *baseImageObject*

restore_wcs()

ACS ImageObjects

Class used to model ACS specific instrument data.

Authors
Christopher Hanley, Warren Hack, Ivo Busko, David Grumm

License
License

class drizzlepac.acsData.**ACSInputImage**(*filename=None*, *output=None*, *group=None*)

Bases: *imageObject*

SEPARATOR = '_'

doUnitConversions()

getdarkcurrent(*extver*)

Return the dark current for the ACS detector. This value will be contained within an instrument specific keyword. The value in the image header will be converted to units of electrons.

Returns

darkcurrent: float
Dark current value for the ACS detector in **units of electrons**.

class drizzlepac.acsData.**WFCInputImage**(*filename=None*, *output=None*, *group=None*)

Bases: *ACSInputImage*

setInstrumentParameters(*instrpars*)

This method overrides the superclass to set default values into the parameter dictionary, in case empty entries are provided.

This method gets called from processInput.

class drizzlepac.acsData.**HRCInputImage**(*filename=None*, *output=None*, *group=None*)

Bases: *ACSInputImage*

setInstrumentParameters(*instrpars*)

This method overrides the superclass to set default values into the parameter dictionary, in case empty entries are provided.

This method gets called from processInput.

class drizzlepac.acsData.**SBCInputImage**(*filename=None*, *output=None*, *group=None*)

Bases: *ACSInputImage*

setInstrumentParameters(*instrpars*)

Sets the instrument parameters.

WFC3 ImageObjects

wfc3Data module provides classes used to import WFC3 specific instrument data.

Authors

Megan Sosey, Christopher Hanley

License

License

class drizzlepac.wfc3Data.**WFC3InputImage**(*filename=None*, *output=None*, *group=None*)

Bases: *imageObject*

SEPARATOR = '_'

class drizzlepac.wfc3Data.**WFC3UVISInputImage**(*filename=None*, *output=None*, *group=None*)

Bases: *WFC3InputImage*

doUnitConversions()

getdarkcurrent(*chip*)

Return the dark current for the WFC3 UVIS detector. This value will be contained within an instrument specific keyword.

Returns

darkcurrent: float

The dark current value with **units of electrons**.

setInstrumentParameters(*instrpars*)

This method overrides the superclass to set default values into the parameter dictionary, in case empty entries are provided.

class drizzlepac.wfc3Data.**WFC3IRInputImage**(*filename=None*, *output=None*, *group=None*)

Bases: *WFC3InputImage*

doUnitConversions()

WF3 IR data come out in electrons, and I imagine the photometry keywords will be calculated as such, so no image manipulation needs be done between native and electrons

getdarkcurrent()

Return the dark current for the WFC3/IR detector. This value will be contained within an instrument specific keyword.

Returns

darkcurrent: float
The dark current value in **units of electrons**.

getdarkimg(*chip*)

Return an array representing the dark image for the detector.

Returns

dark: array
Dark image array in the same shape as the input image with **units of cps**

getexptimeimg(*chip*)

Return an array representing the exposure time per pixel for the detector.

Returns

dark: array
Exposure time array in the same shape as the input image

getskyimg(*chip*)

Notes

Return an array representing the sky image for the detector. The value of the sky is what would actually be subtracted from the exposure by the skysub step.

Units
electrons

setInstrumentParameters(*instrpars*)

This method overrides the superclass to set default values into the parameter dictionary, in case empty entries are provided.

STIS ImageObjects

`stisData` module provides classes used to import STIS specific instrument data.

Authors
Megan Sosey, Christopher Hanley, Mihai Cara

License
License

class drizzlepac.stisData.**STISInputImage**(*filename=None*, *output=None*, *group=None*)

Bases: *imageObject*

SEPARATOR = '_'

doUnitConversions()

Convert the data to electrons.

This converts all science data extensions and saves the results back to disk. We need to make sure the data inside the chips already in memory is altered as well.

getflat(*chip*)

Method for retrieving a detector's flat field. For STIS there are three. This method will return an array the same shape as the image.

class drizzlepac.stisData.**CCDInputImage**(*filename=None*, *output=None*, *group=None*)

Bases: *STISInputImage*

getReadNoise()

Method for returning the readnoise of a detector (in DN).

Units
DN

This should work on a chip, since different chips to be consistant with other detector classes where different chips have different gains.

getdarkcurrent()

Returns the dark current for the STIS CCD chip.

Returns

darkcurrent
[float] Dark current value in **units of electrons** (or counts, if proc_unit=='native').

setInstrumentParameters(*instrpars*)

This method overrides the superclass to set default values into the parameter dictionary, in case empty entries are provided.

class drizzlepac.stisData.**NUVInputImage**(*filename*, *output=None*, *group=None*)

Bases: *STISInputImage*

doUnitConversions()

Convert the data to electrons.

This converts all science data extensions and saves the results back to disk. We need to make sure the data inside the chips already in memory is altered as well.

getdarkcurrent()

Returns the dark current for the STIS NUV detector.

Returns

darkcurrent
[float] Dark current value in **units of electrons** (or counts, if proc_unit==’native’).

setInstrumentParameters(*instrpars*)

This method overrides the superclass to set default values into the parameter dictionary, in case empty entries are provided.

class drizzlepac.stisData.**FUVInputImage**(*filename=None*, *output=None*, *group=None*)

Bases: *STISInputImage*

doUnitConversions()

Convert the data to electrons.

This converts all science data extensions and saves the results back to disk. We need to make sure the data inside the chips already in memory is altered as well.

getdarkcurrent()

Returns the dark current for the STIS FUV detector.

Returns

darkcurrent
[float] Dark current value in **units of electrons** (or counts, if proc_unit==’native’).

setInstrumentParameters(*instrpars*)

This method overrides the superclass to set default values into the parameter dictionary, in case empty entries are provided.

NICMOS ImageObjects

Class used to model NICMOS specific instrument data.

Authors
Christopher Hanley, David Grumm, Megan Sosey

License
License

class drizzlepac.nicmosData.**NICMOSInputImage**(*filename=None*, *output=None*)

Bases: *imageObject*

SEPARATOR = '_'

doUnitConversions()

Convert the data to electrons

This converts all science data extensions and saves the results back to disk. We need to make sure the data inside the chips already in memory is altered as well.

getdarkcurrent()

Return the dark current for the NICMOS detectors.

Returns

darkcurrent
[float] Dark current value with **units of cps**.

getdarkimg(*chip*)

Return an array representing the dark image for the detector.

Returns

dark
[array] The dark array in the same shape as the image with **units of cps**.

getexptimeimg(*chip*)

Return an array representing the exposure time per pixel for the detector.

Returns

dark: array
Exposure time array in the same shape as the input image

getflat(*chip*)

Method for retrieving a detector's flat field.

Returns

flat
[array] The flat field array in the same shape as the input image with **units of cps**.

isCountRate()

isCountRate: Method or IRInputObject used to indicate if the science data is in units of counts or count rate. This method assumes that the keyword 'BUNIT' is in the header of the input FITS file.

class drizzlepac.nicmosData.**NIC1InputImage**(*filename=None*, *output=None*)

Bases: *NICMOSInputImage*

setInstrumentParameters(*instrpars*)

This method overrides the superclass to set default values into the parameter dictionary, in case empty entries are provided.

class drizzlepac.nicmosData.**NIC2InputImage**(*filename=None*, *output=None*)

Bases: *NICMOSInputImage*

createHoleMask()

Add in a mask for the coronographic hole to the general static pixel mask.

setInstrumentParameters(*instrpars*)

This method overrides the superclass to set default values into the parameter dictionary, in case empty entries are provided.

class drizzlepac.nicmosData.**NIC3InputImage**(*filename=None*, *output=None*)

Bases: *NICMOSInputImage*

setInstrumentParameters(*instrpars*)

This method overrides the superclass to set default values into the parameter dictionary, in case empty entries are provided.

class drizzlepac.nicmosData.**darkobject**

darkobject: This class takes as input a pyfits hdulist object. The linear dark and amp glow noise componenets are then extracted from the hdulist.

getampglow()

getampglow: darkobject method which us used to return the amp glow component from a NICMOS temperature dependent dark file.

getampglowheader()

getampglowheader: darkobject method used to return the header information of the amp glow entension of a TDD file.

getlindark()

getlindark: darkobject method which is used to return the linear dark component from a NICMOS temperature dependent dark file.

getlindarkheader()

getlindarkheader: darkobject method used to return the header information of the linear dark entension of a TDD file.

WFPC2 ImageObjects

wfpc2Data module provides classes used to import WFPC2 specific instrument data.

Authors

Warren Hack, Ivo Busko, Christopher Hanley

License

License

class drizzlepac.wfpc2Data.**WFPC2InputImage**(*filename*, *output=None*, *group=None*)

Bases: *imageObject*

SEPARATOR = '_'

buildMask(*chip*, *bits=0*, *write=False*)

Build masks as specified in the user parameters found in the configObj object.

doUnitConversions()

Apply unit conversions to all the chips, ignoring the group parameter. This insures that all the chips get the same conversions when this gets done, even if only 1 chip was specified to be processed.

find_DQ_extension()

Return the suffix for the data quality extension and the name of the file which that DQ extension should be read from.

flat_file_map = {}

getEffGain()

Method used to return the effective gain of a instrument's detector.

Returns

gain
[float] The effective gain.

getReadNoise(*exten*)

Method for returning the readnoise of a detector (in counts).

Returns

readnoise
[float] The readnoise of the detector in **units of counts/electrons**.

getdarkcurrent(*exten*)

Return the dark current for the WFPC2 detector. This value will be contained within an instrument specific keyword. The value in the image header will be converted to units of electrons.

Returns

darkcurrent
[float] Dark current for the WFPC3 detector in **units of counts/electrons**.

getflat(*chip*, *flat_file=None*, *flat_ext=None*)

Method for retrieving a detector's flat field.

Parameters

chip
[int] Chip number. Same as FITS EXTVER.

flat_file
[str, None] Flat field file name. If not specified, it will be determined automatically from image header.

flat_ext
[str, None] Flat field extension name (same as FITS `EXTNAME`). Specifies extension name containing flat field data.

Returns

flat
[numpy.ndarray] The flat-field array in the same shape as the input image.

`setInstrumentParameters`(*instrpars*)

This method overrides the superclass to set default values into the parameter dictionary, in case empty entries are provided.

3.2.5 HAP Glossary

***a priori* WCS**
A WCS derived using pre-existing information about how the observation was taken. The primary example of an *a priori* WCS would be the **GSC240** WCS solutions from the astrometry database which were derived by correcting the coordinates that were used for guiding the observation with updated coordinates for those guide stars derived from the GAIA catalog.

***a posteriori* WCS**
A WCS derived using information measured in the observations themselves and fit to some external reference. A primary example would be the **HSC30** WCS solutions from the astrometry database. These were computed using the Hubble Source Catalog (HSC) positions of sources from the observation, then cross-matching and fitting them to sources from the GAIA catalog. Another example would be the **-FIT** WCS solutions computed during standard pipeline calibration processing. These solutions get derived by identifying sources from each exposure, then cross-matching those sources with GAIA catalog sources and applying the fit to the WCS.

single visit
This term refers to the set of observations taken by a single instrument using the same guide stars potentially spanning multiple orbits. All observations taken by all detectors of that instrument which were requested by the PI in the original proposal are included as this 'single visit' set.

single-visit mosaic (SVM)
A set of drizzled products for all the observations taken in a single visit. All observations in this set of products should be aligned to each other with the exposures for each detector/filter combination being combined into it's own product. All products have the same WCS definition and therefore the same pixel grid to enable direct pixel-by-pixel comparison of the data across all the detectors/filters.

singleton
A single exposure taken as part of a visit that is not part of any pre-defined association based on the proposal.

association
A group of exposures taken together in the same visit, typically defined as a single defined observation in the proposal. For example, a proposal may specify use of a DITHER-LINE pattern and request that it be used to take 15second observations with the F555W filter. All those exposures would be 'associated' by the proposal and result in defining an association table that specifies the names of each of the 15second exposures and the name of the image created by combining all those exposures.

association table
A FITS table listing the filenames of all the input exposures that should be combined together, along with the name or names of the products to be created by combining the input images.

FLT/FLC image
This term refers to the pipeline calibrated version of the input exposures. These files have either **_flt.fits(FLT)** or **_flc.fits(FLC)** in their filename, thus the term FLT/FLC. The FLC files are the CTE-corrected versions of the FLT files, while both copies have identical WCS solutions.

Active WCS
Primary WCS
The WCS solution in the same image header as the science array which defines the transformation from pixel coordinates to world coordinates. The set of keywords that make up this WCS are:

```
* CRVAL1, CRVAL2
* CRPIX1, CRPIX2
* CD1_1, CD1_2, CD2_1, CD2_2
* CTYPE1, CTYPE2
* A_\*_\*, B_\*_\*
```

Alternate WCS
This term refers to any set of WCS keywords which have a single alphabetic character (A-Z) appended to the end, such as CRVAL1A, as defined by FITS WCS Paper I.

CHAPTER

FOUR

USER DATA REPROCESSING

Here we provide resources (including Jupyter Notebooks) to aid users in the reprocessing of their data to best suit their individual science cases.

4.1 Interactive Image Alignment Tools

4.1.1 Image Registration Tasks

A number of tasks have been developed to support the registration of images. This includes documentation for the replacement task for IRAF's `tweakshifts`, currently named `TweakReg`, along with tasks for updating the WCS in HST images and performing photometry equalization for WFPC2 data.

These pages describe how to run the new `TEAL`-enabled task, as well as use the classes in the task to generate catalogs interactively for any chip and work with that catalog. The current implementation of this code relies on a very basic source finding algorithm loosely patterned after the DAOFIND algorithm and does not provide all the same features or outputs found in DAOFIND. The fitting algorithm also reproduces the fitting performed by IRAF's `geomap` in a limited fashion; primarily, it only performs fits equivalent to `geomap`'s 'shift' and 'rscale' solutions. These algorithms will be upgraded as soon as replacements are available.

TWEAKREG

TWEAKREG: Image Alignment

Combining images using `AstroDrizzle` requires that the WCS information in the headers of each input image align to within sub-pixel accuracy. The `TweakReg` task allows the user to align sets of images to each other and/or to and external astrometric reference frame or image. `TweakReg` - A replacement for IRAF-based `tweakshifts`

Authors
Warren Hack, Mihai Cara

License
License

`drizzlepac.tweakreg.`**`TweakReg`**(*files=None*, *editpars=False*, *configobj=None*, *imagefindcfg=None*, *refimagefindcfg=None*, ***input_dict*)

Tweakreg provides an automated interface for computing residual shifts between input exposures being combined using `AstroDrizzle`. The offsets computed by Tweakreg correspond to pointing differences after applying the WCS information from the input image's headers. Such errors would, for example, be due to errors in guide-star positions when combining observations from different observing visits or from slight offsets introduced upon re-acquiring the guide stars in a slightly different position.

Parameters

file
[str or list of str (Default = `'*flt.fits'`)] Input files (passed in from *files* parameter) This paramater can be provided in any of several forms:

- filename of a single image
- filename of an association (ASN)table
- wild-card specification for files in directory (using *, ? etc.)
- comma-separated list of filenames
- `@file` filelist containing list of desired input filenames with one filename on each line of the file.

editpars
[bool (Default = False)] A parameter that allows user to edit input parameters by hand in the GUI. `True` to use the GUI to edit parameters.

configobj
[ConfigObjPars, ConfigObj, dict (Default = None)] An instance of `stsci.tools.cfgpars.ConfigObjPars` or `stsci.tools.configobj.ConfigObj` which overrides default parameter settings. When `configobj` is `defaults`, default parameter values are loaded from the user local configuration file usually located in `~/.teal/tweakreg.cfg` or a matching configuration file in the current directory. This configuration file stores most recent settings that an user used when running `TweakReg` through the TEAL interface. When `configobj` is `None`, `TweakReg` parameters not provided explicitly will be initialized with their default values as described in the "Other Parameters" section.

imagefindcfg
[dict, configObject (Default = None)] An instance of `dict` or `configObject` which overrides default source finding (for input images) parameter settings. See help for `imagefindpars` PSET for a list of available parameters. **Only** the parameters that are different from default values **need** to be specified here.

refimagefindcfg
[dict, configObject (Default = None)] An instance of `dict` or `configObject` which overrides default source finding (for input reference image) parameter settings. See help for `refimagefindpars` PSET for a list of available parameters. **Only** the parameters that are different from default values **need** to be specified here.

input_dict

[dict, optional] An optional list of parameters specified by the user, which can also be used to override the defaults.

Note: This list of parameters **can** include the `updatewcs` parameter, even though this parameter no longer can be set through the TEAL GUI.

Note: This list of parameters **can** contain parameters specific to the `TweakReg` task itself described here in the "Other Parameters" section and **may not** contain parameters from the `refimagefindpars` PSET.

Note: For compatibility purpose with previous `TweakReg` versions, `input_dict` may contain parameters from the the `imagefindpars` PSET. However, if `imagefindcfg` is not `None`, then `imagefindpars` parameters specified through `input_dict` may not duplicate parameters specified through `imagefindcfg`.

Other Parameters

refimage

[str (Default = '')] Filename of reference image. Sources derived from this image will be used as the reference for matching with sources from all input images unless a separate catalog is provided through the `refcat` parameter. In addition, this image file must contain a valid not distorted WCS that will define the projection plane in which image alignment is performed ("reference WCS"). When `refimage` is not provided, a reference WCS will be derived from input images.

expand_refcat

[bool (Default = False)] Specifies whether to add new sources from just matched images to the reference catalog to allow next image to be matched against an expanded reference catalog.

enforce_user_order

[bool (Default = True)] Specifies whether images should be aligned in the order specified in the `file` input parameter or `TweakReg` should optimize the order of alignment by intersection area of the images. Default value (`True`) will align images in the user specified order, except when some images cannot be aligned in which case `TweakReg` will optimize the image alignment order. Alignment order optimization is available *only* when `expand_refcat` = `True`.

exclusions: string (Default = '')

This parameter allows the user to specify a set of files which contain regions in the image to ignore (or to include) when finding sources. This file MUST have 1 line for each input image with the name of the input file in the first column. Subsequent columns would be used to specify an inclusion and/or exclusion file for each chip in `'SCI,<n>'` index order. If a chip does not require an exclusion file, the string

None or INDEF can be used as a placeholder for that chip. Each exclusion file can be either a mask provided as a simple FITS file or a region file in DS9-format.

When a mask file is provided, TweakReg will look for the first image-like extension with image data of the same dimensions as the input image. Zeros in the mask will be interpreted as "bad" (excluded from search) pixels while non-zero pixels will be interpreted as "good" pixels. It is recommended that mask files be FITS files without extensions and mask data (preferably of integer type) reside in the primary HDU.

If a region file is provided then it should conform to the 'region' file format generated by DS9. The region files can contain both regular ("include") regions as well as "exclude" regions. Regular ("include") regions indicate the regions of the image that should be searched for sources while "exclude" regions indicate parts of the image that should not be used for source detection. The "ruler", "compass", and "projection" regions are not supported (ignored). When **all regions** in a region file are "exclude" regions, then it will be assumed that the entire image is "good" before the exclude regions are processed. In other words, an "include" region corresponding to the entire image will be *prepended* to the list of exclude regions.

Note: Regions in a region file are processed in the order they appear in the region file. Thus, when region files contain *both* "include" and "exclude" regions, the order in which these regions appear may affect the results.

Warning: TweakReg relies on pyregion package for work with region files. At the time of writing, pyregion uses a different algorithm from DS9 for converting regions from sky coordinates to image coordinate (this conversion is performed before regions are converted to masks). For these reasons, regions provided in sky coordinates may not produce the expected (from DS9) results. While in most instances these discrepancies should be tolerable, it is important to keep this in mind.

During testing it was observed that conversion to image coordinates is most accurate for polygonal regions and less accurate for other regions Therefore, if one must provide regions in sky coordinates, it is types. recommended to use polygonal and circular regions and to avoid elliptical and rectangular regions as their conversion to image coordinates is less accurate. One may use mapreg task in the drizzlepac package to convert region files from sky coordinates to image coordinates. This will allow one to see the actual regions that will be used by source finding routine in TweakReg.

updatewcs

[bool (Default = No)] **NOT available through TEAL GUI interface.** This parameter can only be set through the Python interface to Tweakreg by passing it in as part of the input_dict in order to insure that running updatewcs **does not**

overwrite a previously determined solution written out to the input file headers.

writecat
[bool (Default = Yes)] Specify whether or not to write out the source catalogs generated for each input image by the built-in source extraction algorithm.

clean
[bool (Default = No)] Specify whether or not to remove the temporary files created by `TweakReg`, including any catalog files generated for the shift determination.

interactive
[bool (Default = Yes)] This switch controls whether the program stops and waits for the user to examine any generated plots before continuing on to the next image. If turned off, plots will still be displayed, but they will also be saved to disk automatically as a PNG image with an autogenerated name without requiring any user input.

verbose
[bool (Default = No)] Specify whether or not to print extra messages during processing.

runfile
[string (Default = 'tweakreg.log')] Specify the filename of the processing log.

UPDATE HEADER
updatehdr
[bool (Default = No)] Specify whether or not to update the headers of each input image directly with the shifts that were determined. This will allow the input images to be combined by `AstroDrizzle` without having to provide the shiftfile as well.

wcsname
[str (Default = 'TWEAK')] Name of updated primary WCS.

reusename
[bool (Default = False)] Allows overwriting of an existing primary WCS with the same name as specified by `wcsname` parameter.

HEADERLET CREATION
headerlet: bool (Default = No)
Specify whether or not to generate a headerlet from the images at the end of the task? If turned on, this will create a headerlet from the images regardless of the value of the `updatehdr` parameter.

attach: bool (Default = Yes)
If creating a headerlet, choose whether or not to attach the new headerlet to the input image as a new extension.

hdrfile: string (Default = '')
Filename to use for writing out headerlet to a separate file. If the name does not contain `.fits`, it will create a filename from the rootname of the input image, the value of this string, and it will end in `'_hlet.fits'`. For example, if only `'hdrlet1'` is given, the full filename created will

be 'j99da1f2q_hdrlet1_hlet.fits' when creating a headerlet for image 'j99da1f2q_flt.fits'.

clobber: bool (Default = No)
If a headerlet with 'hdrfile' already exists on disk, specify whether or not to overwrite that previous file.

hdrname: string (Default = '')
Unique name to give to headerlet solution. This name will be used to identify this specific WCS alignment solution contained in the headerlet.

author: string, optional (Default = '')
Name of the creator of the headerlet.

descrip: string, optional (Default = '')
Short (1-line) description to be included in headerlet as DESCRIP keyword. This can be used to provide a quick look description of the WCS alignment contained in the headerlet.

catalog: string, optional (Default = '')
Name of reference catalog used as the basis for the image alignment.

history: string, optional (Default = '')
Filename of a file containing detailed information regarding the history of the WCS solution contained in the headerlet. This can include information on the catalog used for the alignment, or notes on processing that went into finalizing the WCS alignment stored in this headerlet. This information will be reformatted as 70-character wide FITS HISTORY keyword section.

OPTIONAL SHIFTFILE OUTPUT
shiftfile
[bool (Default = No)] Create output shiftfile?

outshifts
[str (Default = 'shifts.txt')] The name for the output shift file created by TweakReg. This shiftfile will be formatted for use as direct input to AstroDrizzle.

outwcs
[str (Default = 'shifts_wcs.fits')] Filename to be given to the OUTPUT reference WCS file created by TweakReg. This reference WCS defines the WCS from which the shifts get measured, and will be used by AstroDrizzle to interpret those shifts. This reference WCS file will be a FITS file that only contains the WCS keywords in a Primary header with no image data itself. The values will be derived from the FIRST input image specified.

COORDINATE FILE DESCRIPTION
catfile
[str (Default = '')] Name of file that contains a list of input images and associated catalog files generated by the user. Each line of this file will contain the name of an input image in the first column. The remaining columns will provide the names of the source catalogs for each chip in order of the science extension numbers ((SCI,1), (SCI,2), ...).

A sample catfile, with one line per image would look like:

```
image1_flt.fts  cat1_sci1.coo  cat1_sci2.coo
image2_flt.fts  cat2_sci1.coo  cat2_sci2.coo
```

Note: Catalog files themselves must be text files containing "white space"-separated list of values (`xcol`, `ycol`, etc.)

xcol
[int (Default = 1)] Column number of X position from the user-generated catalog files specified in the catfile.

ycol
[int (Default = 2)] Column number of Y position from the user-generated catalog files specified in the catfile.

fluxcol
[int (Default = None)] Column number for the flux values from the user-generated catalog files specified in the catfile. These values will only be used if a flux limit has been specified by the user using the `maxflux` or `minflux` parameters.

maxflux
[float (Default = None)] Limiting flux value for selecting valid objects in the input image's catalog. If specified, this flux will serve as the upper limit of a range for selecting objects to be used in matching with objects identified in the reference image. If the value is set to `None`, all objects with fluxes brighter than the minimum specified in `minflux` will be used. If both values are set to `None`, all objects will be used.

minflux
[float (Default = None)] Limiting flux value for selecting valid objects in the input image's catalog. If specified, this flux value will serve as the lower limit of a range for selecting objects to be used in matching with objects identified in the reference image. If the value is set to `None`, all objects fainter than the limit specified by `maxflux` will be used. If both values are set to `None`, all objects will be used.

fluxunits
[str {'counts', 'cps', 'mag'} (Default = 'counts')] This allows the task to correctly interpret the flux limits specified by `maxflux` and `minflux` when sorting the object list for trimming of fainter objects.

xyunits
[str {'pixels', 'degrees'} (Default = 'pixels')] Specifies whether the positions in this catalog are already sky pixel positions, or whether they need to be transformed to the sky.

nbright
[int (Default = None)] The number of brightest objects to keep after sorting the full object list. If nbright is set equal to `None`, all objects will be used.

REFERENCE CATALOG DESCRIPTION

refcat

[str (Default = '')] Name of the external reference catalog file to be used in place of the catalog extracted from one of the input images. When `refimage` is not specified, reference WCS to be used with reference catalog will be derived from input images.

Note: Reference catalog must be text file containing "white space"-separated list of values (`xcol`, `ycol`, etc.)

---

refxcol

[int (Default = 1)] Column number of RA in the external catalog file specified by the refcat.

refycol

[int (Default = 2)] Column number of Dec in the external catalog file specified by the refcat.

refxyunits

[str {'pixels','degrees'} (Default = 'degrees')] Units of sky positions.

rfluxcol

[int (Default = None)] Column number of flux/magnitude values in the external catalog file specified by the refcat.

rmaxflux

[float (Default = None)] Limiting flux value used to select valid objects in the external catalog. If specified, the flux value will serve as the upper limit of a range for selecting objects to be used in matching with objects identified in the reference image. If the value is set to `None`, all objects with fluxes brighter than the minimum specified in `rminflux` will be used. If both values are set to `None`, all objects will be used.

rminflux

[float (Default = None)] Limiting flux value used to select valid objects in the external catalog. If specified, the flux will serve as the lower limit of a range for selecting objects to be used in matching with objects identified in the reference image. If the value is set to `None`, all objects fainter than the limit specified by `rmaxflux` will be used. If both values are set to `None`, all objects will be used.

rfluxunits

[{'counts', 'cps', 'mag'} (Default = 'mag')] This allows the task to correctly interpret the flux limits specified by `rmaxflux` and `rminflux` when sorting the object list for trimming of fainter objects.

refnbright

[int (Default = None)] Number of brightest objects to keep after sorting the full object list. If refnbright is set to `None`, all objects will be used. Used in conjunction with refcat.

OBJECT MATCHING PARAMETERS

minobj
[int (Default = 15)] Minimum number of identified objects from each input image to use in matching objects from other images.

searchrad
[float (Default = 1.0)] The search radius for a match.

searchunits
[str (Default = 'arcseconds')] Units for search radius.

use2dhist
[bool (Default = Yes)] Use 2d histogram to find initial offset?

see2dplot
[bool (Default = Yes)] See 2d histogram for initial offset?

tolerance
[float (Default = 1.0)] The matching tolerance in pixels after applying an initial solution derived from the 'triangles' algorithm. This parameter gets passed directly to `xyxymatch` for use in matching the object lists from each image with the reference image's object list.

separation
[float (Default = 0.0)] The minimum separation for objects in the input and reference coordinate lists. Objects closer together than 'separation' pixels are removed from the input and reference coordinate lists prior to matching. This parameter gets passed directly to `xyxymatch` for use in matching the object lists from each image with the reference image's object list.

xoffset
[float (Default = 0.0)] Initial estimate for the offset in X between the images and the reference frame. This offset will be used for all input images provided. If the parameter value is set to `None`, no offset will be assumed in matching sources in `xyxymatch`.

yoffset
[float (Default = 0.0)] Initial estimate for the offset in Y between the images and the reference frame. This offset will be used for all input images provided.If the parameter value is set to None, no offset will be assumed in matching sources in `xyxymatch`.

CATALOG FITTING PARAMETERS

fitgeometry
[str {'shift', 'rscale', 'general'} (Default = 'rscale')] The fitting geometry to be used in fitting the matched object lists. This parameter is used in fitting the offsets, rotations and/or scale changes from the matched object lists. The 'general' fit geometry allows for independent scale and rotation for each axis.

residplot
[str {'No plot', 'vector', 'residuals', 'both'} (Default = 'both')] Plot residuals from fit? If 'both' is selected, the 'vector' and 'residuals' plots will be displayed in separate plotting windows at the same time.

nclip
[int (Default = 3)] Number of clipping iterations in fit.

sigma
[float (Default = 3.0)] Clipping limit in sigma units.

ADVANCED PARAMETERS AVAILABLE FROM COMMAND LINE
updatewcs
[bool (Default = No)] This parameter specifies whether the WCS keywords are to be updated by running updatewcs on the input data, or left alone. The update performed by updatewcs not only recomputes the WCS based on the currently used `IDCTAB`, but also populates the header with the `SIP` coefficients. For `ACS/WFC` images, the time-dependence correction will also be applied to the `WCS` and `SIP` keywords. This parameter should be set to 'No' (`False`) when the WCS keywords have been carefully set by some other method, and need to be passed through to drizzle 'as is', otherwise those updates will be over-written by this update.

Note: This parameter was preserved in the API for compatibility purposes with existing user processing pipe-lines. However, it has been removed from the `TEAL` interface because it is easy to have it set to 'yes' (especially between consecutive runs of `AstroDrizzle`) with potentially disastrous effects on input image WCS (for example it could wipe-out previously aligned WCS).

See also:

drizzlepac.astrodrizzle

Notes

Tweakreg supports the use of calibrated, distorted images (such as FLT images for ACS and WFC3, or `_c0m.fits` images for WFPC2) as input images. All coordinates for sources derived from these images (either by this task or as provided by the user directly) will be corrected for distortion using the distortion model information specified in each image's header. This eliminates the need to run `AstroDrizzle` on the input images prior to running `TweakReg`.

Note: All calibrated input images must have been updated using `updatewcs` from the `STWCS` package, to include the full distortion model in the header. Alternatively, one can set `updatewcs` parameter to `True` when running either `TweakReg` or `AstroDrizzle` from command line (Python interpreter) **the first time** on such images.

This task will use catalogs, and catalog-matching, based on the `xyxymatch` algorithm to determine the offset between the input images. The primary mode of operation will be to extract a catalog of source positions from each input image using either a 'DAOFIND-like' algorithm or `SExtractor` (if the user has `SExtractor` installed). Alternatively, the user can provide their catalogs of source positions derived from **each input chip**.

Note: Catalog files must be text files containing "white space"-separated list of values (`xcol`, `ycol`, etc.)

The reference frame will be defined either by:

- the image with the largest overlap with another input image AND with the largest total overlap with the rest of the input images,
- a catalog derived from a reference image specified by the user, or
- a catalog of undistorted sky positions (RA/Dec) and fluxes provided by the user.

For a given observation, the distortion model is applied to all distorted input positions, and the sources from each chip are then combined into a single catalog of undistorted positions.

The undistorted positions for each observation then get passed to `xyxymatch` for matching to objects from the reference catalog.

The source lists from each image will generally include cosmic-rays as detected sources, which can at times significantly confuse object identification between images. Observations that include long exposures often have more cosmic-ray events than source objects. As such, isolating the cosmic-ray events in those cases would significantly improve the efficiency of common source identification between images. One such method for trimming potential false detections from each source list would be to set a flux limit to exclude detections below that limit. As the fluxes reported in the default source object lists are provided as magnitude values, setting the `maxflux` or `minflux` parameter value to a magnitude- based limit, and then setting the `ascend` parameter to `True`, will allow for the creations of catalogs trimmed of all sources fainter than the provided limit. The trimmed source list can then be used in matching sources between images and in establishing the final fitting for the shifts.

A fit can then be performed on the matched set of positions between the input and the reference to produce the 'shiftfile'. If the user is confident that the solution will be correct, the header of each input image can be updated directly with the fit derived for that image. Otherwise, the 'shiftfile' can be passed to AstroDrizzle for aligning the images.

Note: Because of the nature of the used algorithm it may be necessary to run this task multiple time until new shifts, rotations, and/or scales are small enough for the required precision.

New sources (that are not in the reference catalog) from the matched images are added to the reference catalog in order to allow next image to be matched to a larger reference catalog. This allows alignment of images that do not overlap directly with the reference image and/or catalog and it is particularly useful in image registration of large mosaics. Addition of new sources to the reference catalog can be turned off by setting `expand_refcat` to `False` when using an external reference catalog. When an external catalog is not provided (`refcat``=''`) or when using an external reference catalog with ``expand_refcat` set to `True` (assuming `writecat` = `True` and `clean` = `False`), the list of all sources in the expanded reference catalog is saved in a catalog file named `cumulative_sky_refcat_###.coo` where ### is the base file name derived from either the external catalog (if provided) or the name of the image used as the reference image.

When `enforce_user_order` is `False`, image catalogs are matched to the reference catalog in order

of decreasing overlap area with the reference catalog, otherwise user order of files specified in the `file` parameter is used.

Format of Exclusion Catalog

The format for the exclusions catalog requires 1 line in the file for every input image, regardless of whether or not that image has any defined exclusion regions. A sample file would look like:

```
j99da1emq_flt.fits
j99da1f2q_flt.fits test_exclusion.reg
```

This file specifies no exclusion files for the first image, and only an regions file for SCI,1 of the second image. NOTE: The first file can be dropped completely from the exclusion catalog file.

In the above example, should an exclusion regions file only be needed for the second chip in the second image, the file would need to look like:

```
j99da1emq_flt.fits
j99da1f2q_flt.fits None test_sci2_exclusion.reg
```

The value `None` could also be replaced by `INDEF` if desired, but either string needs to be present to signify no regions file for that chip while the code continues parsing the line to find a file for the second chip.

Format of Region Files

The format of the exclusions catalogs referenced in the 'exclusions' file defaults to the format written out by DS9 using the 'DS9/Funtools' region file format. A sample file with circle() regions will look like:

```
# Region file format: DS9 version 4.1
# Filename: j99da1f2q_flt.fits[SCI]
global color=green dashlist=8 3 width=1 font="helvetica 10 normal roman"
select=1 highlite=1 dash=0 fixed=0 edit=1 move=1 delete=1 include=1 source=1
image
circle(3170,198,20)
ellipse(3269,428,30,10,45) # a rotated ellipse
box(3241.1146,219.78132,20,20,15) # a rotated box
circle(200,200,50)  # outer circle
-circle(200,200,30) # inner circle
```

This region file will be interpreted as "find all sources in the image that **are inside** the four regions above but **not inside** the region -circle(200,200,30)". Effectively we will instruct `TweakReg` to find all the sources *inside* the following regions:

```
circle(3170,198,20)
ellipse(3269,428,30,10,45) # a rotated ellipse
box(3241.1146,219.78132,20,20,15) # a rotated box
annulus(200,200,30,50)  # outer circle(r=50) - inner circle(r=30)
```

Examples

The tweakreg task can be run from either the TEAL GUI or from the command-line using Python. These examples illustrate the various syntax options available.

Example 1: Align a set of calibrated (`_flt.fits`) images using `IMAGEFIND`, a built-in source finding algorithm based on `DAOPHOT`. Auto-detect the sky sigma value and select sources > 200 sigma. (Auto-sigma is computed from the first input exposure as: `1.5*imstat(image,nclip=3, fields='stddev')`.) Set the convolution kernel width to ~2x the value of the PSF FWHM. Save the residual offsets (`dx`, `dy`, `rot`, `scale`, `xfit_rms`, `yfit_rms`) to a text file.

1. Run the task from Python using the command line while individually specifying source finding parameters for the reference image and input images:

```
>>> import drizzlepac
>>> from drizzlepac import tweakreg
>>> tweakreg.TweakReg('*flt.fits',
...       imagefindcfg={'threshold' : 200, 'conv_width' : 3.5},
...       refimagefindcfg={'threshold' : 400, 'conv_width' : 2.5},
...       updatehdr=False, shiftfile=True, outshifts='shift.txt')
```

or, using `dict` constructor,

```
>>> import drizzlepac
>>> from drizzlepac import tweakreg
>>> tweakreg.TweakReg('*flt.fits',
...       imagefindcfg=dict(threshold=200, conv_width=3.5),
...       refimagefindcfg=dict(threshold=400, conv_width=2.5),
...       updatehdr=False, shiftfile=True, outshifts='shift.txt')
```

Or, run the same task from the Python command line, but specify all parameters in a config file named "myparam.cfg":

```
>>> tweakreg.TweakReg('*flt.fits', configobj='myparam.cfg')
```

Alternately, edit the imagefind parameters in a TEAL GUI window prior to running the task:

```
>>> tweakreg.edit_imagefindpars()
```

2. Help can be accessed via the "Help" pulldown menu in the TEAL GUI. It can also be accessed from the Python command-line and saved to a text file:

```
>>> from drizzlepac import tweakreg
>>> tweakreg.help()
```

or

```
>>> tweakreg.help(file='help.txt')
```

drizzlepac.tweakreg.**help**(*file=None*)

Print out syntax help for running `tweakreg`.

Parameters

file

[str (Default = None)] If given, write out help to the filename specified by this parameter Any previously existing file with this name will be deleted before writing out the help.

Imagefindpars: Source finding parameters

This interface provides a mechanism for setting the parameters used by the built-in source finding algorithm based on the published algorithms used by DAOFIND. These parameters control the operation of the algorithm that extracts sources from the image as called by *TWEAKREG: Image Alignment*. The algorithm implemented follows the concepts defined by `DAOFIND` (without actually using any `DAOFIND` code).

Attributes

computesig: bool (Default = True)
This parameter controls whether or not to automatically compute a sigma value to be used for object identification. If set to `True`, then the value computed will override any user input for the parameter `skysigma`. The automatic sigma value gets computed from each input exposure as:

$$\sigma = \sqrt{2\,|mode|}$$

This single value will then be used for object identification for all input exposures.

skysigma: float (Default = 0.0)
The standard deviation of the sky pixels. This value will only be used if computesig is `False`.

conv_width: float (Default = 3.5)
The convolution kernel width in pixels. Recommended values (~2x the PSF FWHM): ACS/WFC & WFC3/UVIS ~3.5 pix and WFC3/IR ~2.5 pix.

peakmin: float, None (Default = None)
This parameter allows the user to select only those sources whose peak value (in the units of the input image) is greater than this value.

peakmax: float, None (Default = None)
This parameter allows the user to select only those sources whose peak value (in the units of the input image) is less than this value.

threshold: float (Default = 4.0)
The object detection threshold above the local background in units of sigma.

nsigma: float (Default = 1.5)
The semi-major axis of the Gaussian convolution kernel used to compute the density enhancement and mean density images in Gaussian sigma.

ratio: float (Default = 1.0)
The ratio of the sigma of the Gaussian convolution kernel along the minor axis direction to the sigma along the major axis direction. For a circularly-symmetric kernel use ratio = 1.0.

theta: float (Default = 0.0)
The position angle (degrees) of the major axis of the Gaussian convolution kernel. Theta is measured counter-clockwise from the x axis.

fluxmin: float, None (Default = None)
This parameter allows the user to select only those sources whose total flux (in the units of the input image) is greater than this value.

fluxmax: float, None (Default = None)
This parameter allows the user to select only those sources whose total flux (in the units of the input image) is less than this value.

dqbits: int, str, None, optional (Default = None)
Integer sum of all the DQ bit values from the input image's DQ array that should be considered "good" when building masks for source finding. For example, if pixels in the DQ array can be combinations of 1, 2, 4, and 8 flags and one wants to consider DQ "defects" having flags 2 and 4 as being acceptable for source finding, then `dqbits` should be set to 2+4=6. Then a DQ pixel having values 2,4, or 6 will be considered a good pixel, while a DQ pixel with a value, e.g., 1+2=3, 4+8=12, etc. will be flagged as a "bad" pixel.

Alternatively, one can enter a comma- or '+'-separated list of integer bit flags that should be added to obtain the final "good" bits. For example, both `4,8` and `4+8` are equivalent to setting `dqbits` to 12.

Setting `dqbits` to 0 will make *all* non-zero pixels in the DQ mask to be considered "bad" pixels, and the corresponding image pixels will not be used for source finding.

The default value of `None` will turn off the use of image's DQ array for source finding.

In order to reverse the meaning of the `dqbits` parameter from indicating values of the "good" DQ flags to indicating the "bad" DQ flags, prepend '~' to the string value. For example, in order not to use pixels with DQ flags 4 and 8 for source finding and to consider as "good" all other pixels (regardless of their DQ flag), set `dqbits` to `~4+8`, or `~4,8`. To obtain the same effect with an `int` input value (except for 0), enter -(4+8+1)=-13. Following this convention, a `dqbits` string value of `'~0'` would be equivalent to setting `dqbits=None`.

use_sharp_round: bool (Default = False)
This parameter controls whether or not to enable selection of sources based on their

sharpness and roundness statistics.

sharplo: float (Default = 0.2)
`sharplo` and `sharphi` are designed to eliminate brightness maxima that are due to bad pixels rather than to astronomical objects. Only sources with sharpness above the `sharplo` value will be selected.

sharphi: float (Default = 1.0)
`sharplo` and `sharphi` are designed to eliminate brightness maxima that are due to bad pixels rather than to astronomical objects. Only sources with sharpness below the `sharphi` value will be selected.

roundlo: float (Default = -1.0)
`roundlo` and `roundhi` are designed to eliminate brightness maxima that are due to bad rows or columns, rather than to astronomical objects. Only sources with roundness above the `roundlo` value will be selected.

roundhi: float (Default = 1.0)
`roundlo` and `roundhi` are designed to eliminate brightness maxima that are due to bad rows or columns, rather than to astronomical objects. Only sources with roundness below the `roundhi` value will be selected.

`drizzlepac.imagefindpars.`**`help`**(*file=None*)

Print out syntax help for running `imagefindpars`.

Parameters

file
[str (Default = None)] If given, write out help to the filename specified by this parameter Any previously existing file with this name will be deleted before writing out the help.

Refmagefindpars: Source finding parameters for the reference image

This interface provides a mechanism for setting the parameters used by the built-in source finding algorithm based on the published algorithms used by DAOFIND. These settings apply only to source finding in the reference images. These parameters control the operation of the algorithm that extracts sources from the reference image (if specified) as called by *TWEAKREG: Image Alignment*. The algorithm implemented follows the concepts defined by `DAOFIND` (without actually using any `DAOFIND` code).

Attributes

computesig: bool (Default = True)
This parameter controls whether or not to automatically compute a sigma value to be used for object identification. If set to `True`, then the value computed will override any user input for the parameter `skysigma`. The automatic sigma value gets computed from each input exposure as:

$$\sigma = \sqrt{2\,|mode|}$$

This single value will then be used for object identification for all input exposures.

skysigma: float (Default = 0.0)
The standard deviation of the sky pixels. This value will only be used if computesig is `False`.

conv_width: float (Default = 3.5)
The convolution kernel width in pixels. Recommended values (~2x the PSF FWHM): ACS/WFC & WFC3/UVIS ~3.5 pix and WFC3/IR ~2.5 pix.

peakmin: float, None (Default = None)
This parameter allows the user to select only those sources whose peak value (in the units of the input image) is greater than this value.

peakmax: float, None (Default = None)
This parameter allows the user to select only those sources whose peak value (in the units of the input image) is less than this value.

threshold: float (Default = 4.0)
The object detection threshold above the local background in units of sigma.

nsigma: float (Default = 1.5)
The semi-major axis of the Gaussian convolution kernel used to compute the density enhancement and mean density images in Gaussian sigma.

ratio: float (Default = 1.0)
The ratio of the sigma of the Gaussian convolution kernel along the minor axis direction to the sigma along the major axis direction. For a circularly-symmetric kernel use ratio = 1.0.

theta: float (Default = 0.0)
The position angle (degrees) of the major axis of the Gaussian convolution kernel. Theta is measured counter-clockwise from the x axis.

fluxmin: float, None (Default = None)
This parameter allows the user to select only those sources whose total flux (in the units of the input image) is greater than this value.

fluxmax: float, None (Default = None)
This parameter allows the user to select only those sources whose total flux (in the units of the input image) is less than this value.

dqbits: int, str, None, optional (Default = None)
Integer sum of all the DQ bit values from the input image's DQ array that should be considered "good" when building masks for source finding. For example, if pixels in the DQ array can be combinations of 1, 2, 4, and 8 flags and one wants to consider DQ "defects" having flags 2 and 4 as being acceptable for source finding, then `dqbits` should be set to 2+4=6. Then a DQ pixel having values 2,4, or 6 will be considered a good pixel, while a DQ pixel with a value, e.g., 1+2=3, 4+8=12, etc. will be flagged as a "bad" pixel.

Alternatively, one can enter a comma- or '+'-separated list of integer bit flags that should be added to obtain the final "good" bits. For example, both `4,8` and `4+8` are equivalent to setting `dqbits` to 12.

Setting `dqbits` to 0 will make *all* non-zero pixels in the DQ mask to be considered "bad" pixels, and the corresponding image pixels will not be used for source finding.

The default value of `None` will turn off the use of image's DQ array for source finding.

In order to reverse the meaning of the `dqbits` parameter from indicating values of the "good" DQ flags to indicating the "bad" DQ flags, prepend '~' to the string value. For example, in order not to use pixels with DQ flags 4 and 8 for source finding and to consider as "good" all other pixels (regardless of their DQ flag), set `dqbits` to `~4+8`, or `~4,8`. To obtain the same effect with an `int` input value (except for 0), enter -(4+8+1)=-13. Following this convention, a `dqbits` string value of `'~0'` would be equivalent to setting `dqbits=None`.

use_sharp_round: bool (Default = False)
This parameter controls whether or not to enable selection of sources based on their sharpness and roundness statistics.

sharplo: float (Default = 0.2)
`sharplo` and `sharphi` are designed to eliminate brightness maxima that are due to bad pixels rather than to astronomical objects. Only sources with sharpness above the `sharplo` value will be selected.

sharphi: float (Default = 1.0)
`sharplo` and `sharphi` are designed to eliminate brightness maxima that are due to bad pixels rather than to astronomical objects. Only sources with sharpness below the `sharphi` value will be selected.

roundlo: float (Default = -1.0)
`roundlo` and `roundhi` are designed to eliminate brightness maxima that are due to bad rows or columns, rather than to astronomical objects. Only sources with roundness above the `roundlo` value will be selected.

roundhi: float (Default = 1.0)
`roundlo` and `roundhi` are designed to eliminate brightness maxima that are due to bad rows or columns, rather than to astronomical objects. Only sources with roundness below the `roundhi` value will be selected.

`drizzlepac.refimagefindpars.`**`help`**(*file=None*)

Print out syntax help for running `refimagefindpars`.

Parameters

file
[str (Default = None)] If given, write out help to the filename specified by this parameter Any previously existing file with this name will be deleted before writing out the help.

Region mapping for TweakReg

This module provides functions for mapping DS9 region files given in sky coordinates to DS9 region files specified in image coordinates of multiple images using the WCS information from the images.

Authors
Mihai Cara

License
License

drizzlepac.mapreg.**MapReg**(*input_reg*, *images*, *img_wcs_ext='sci'*, *refimg=''*, *ref_wcs_ext='sci'*, *chip_reg=''*, *outpath='./regions'*, *filter=''*, *catfname=''*, *iteractive=False*, *append=False*, *verbose=True*)

MapReg() provides an automated interface for converting a region file to the image coordinate system (CS) of multiple images (and their extensions) using WCS information from the image(s) header(s). This conversion does not take into account pointing errors and, therefore, an examination and adjustment (if required) of output region files is highly recommended. This task is designed to simplify the creation of the exclusions and/or inclusions region files used with *TweakReg()* task for sources finding.

Parameters

input_reg
[string or list of strings (Default = '')] Input region files that need to be mapped to image CS using WCS information from `images` (see below). Only region files saved in sky CS are allowed in this release. Regions specified in image-like coordinates (e.g., image, physical) will be ignored.

This paramater can be provided in any of several forms:

- filename of a single image
- comma-separated list of filenames
- `@-file` filelist containing list of desired input region filenames

The `@-file` filelist needs to be provided as an ASCII text file containing a list of filenames for all input region files with one filename on each line of the file.

images
[string or list of strings (Default = `*.fits`)] FITS images onto which the region files `input_reg` will be mapped. These image files must contain WCS information in their headers in order to convert `input_reg` from sky coordinates to correct image coordinates. This paramater can be provided in any of several forms:

- filename of a single image
- filename of an association (ASN)table
- wild-card specification for files in directory (using *, ? etc.)
- comma-separated list of filenames

- `@-file` filelist containing list of desired input filenames (and optional inverse variance map filenames)

The `@-file` filelist needs to be provided as an ASCII text file containing a list of filenames for all input images (to which `input_reg` regions should be mapped) with one filename on each line of the file.

img_wcs_ext

[string or list of strings (Default = `SCI`)] Extension name, extension name and version, or extension number of FITS extensions in the `images` to which the input regions `input_reg` should be mapped. The header of each extension must contain WCS information that will be used to convert `input_reg` from sky CS to image-like CS. Multiple extensions must be separated by semicolon while extension name and version (if present) must be separated by comma, e.g., `'SCI;DQ,1;0'`. When specifying the extension name only, internally it will be expanded into a list of extension names and versions for each version of that extension name present in the input `images`. For example, if a FITS file has four SCI and four DQ extensions, then `'SCI;DQ,1;0'` will be expanded into `'SCI,1;SCI,2;SCI,3;SCI,4;DQ,1;0'`.

refimg

[string (Default = '')] **Reserved for future use.** Filename of the reference image. May contain extension specifier: [extname,extver], [extname], or [extnumber].

Note: This parameter is reserved for future use and it is not available through `TEAL` interface.

---

ref_wcs_ext

[string (Default = `SCI`)] **Reserved for future use.** Extension name and/or version of FITS extensions in the `refimg` that contain WCS information that will be used to convert `input_reg` from image-like CS to sky CS. NOTE: Only extension name is allowed when `input_reg` is a list of region files that contain regions in image-like CS. In this case, the number of regions in `input_reg` must agree with the number of extensions with name specified by `ref_wcs_ext` present in the `refimg` FITS image.

Note: This parameter is reserved for future use and it is not available through `TEAL` interface.

---

chip_reg

[string or list of strings (Default = '')] Input region files in image CS associated with each extension specified by the `img_wcs_ext` parameter above. These regions will be added directly (without any transformation) to the `input_reg` regions mapped to each extension of the input `images`. These regions must be specified in image-like coordinates. Typically, these regions should contain "exclude" regions to exclude parts of the image specific to the detector **chip** (e.g., vignetted regions due to used filters, or occulting finger in ACS/HRC images) from being

used for source finding. This paramater can be provided in one of the following forms:

- filename of a single image (if `img_wcs_ext` specifies a single FITS extension);
- comma-separated list of filenames (if `img_wcs_ext` specifies more than one extension) or `None` for extensions that do not need any chip-specific regions to be excluded/included;
- '' (empty string) or None if no chip-specific region files are provided.

The number of regions ideally must be equal to the number of extensions specified by the `img_wcs_ext` parameter. If the number of chip-specific regions is less than the number of `img_wcs_ext` extensions then 'chip_reg' regions will be assigned to the first extensions from `img_wcs_ext` (after internal expansion described in help for the `img_wcs_ext` parameter above). If the number of 'chip_reg' is larger than the number of `img_wcs_ext` extensions then extra regions will be ignored.

outpath
[string (Default = `./regions`)] The directory to which the transformed regions should be saved.

filter
[string { 'None', 'fast', or 'precise' } (Default = 'None')] Specify whether or not to remove the regions in the transformed region files that are outside the image array. With the 'fast' mehod only intersection of the bounding boxes is being checked.

Note: The `'precise'` method is not implemented in this release and, if specified, defaults to 'fast'. The `'precise'` option is not available through the `TEAL` interface.

---

catfname
[string (Default = `exclusions_cat.txt`)] The file name of the output exclusions catalog file to be created from the supplied image and region file names. This file can be passed as an input to TweakReg task. Verify that the created file is correct!

append
[bool (Default = False)] Specify whether or not to append the transformed regions to the existing region files with the same name.

interactive
[bool (Default = False)] **Reserved for future use.** (This switch controls whether the program stops and waits for the user to examine any generated region files before continuing on to the next image.)

Note: This parameter is reserved for future use and it is not available through `TEAL` interface.

---

verbose
: [bool (Default = False)] Specify whether or not to print extra messages during processing.

Notes

NOTE 1: This task takes a region file (or multiple files) that describe(s) what regions of sky should be used for source finding (*include* regions) and what regions should be avoided (*exclude* regions) and transforms/maps this region file onto a number of image files that need to be alighned.

The idea behind this task is automate the creation of region files that then can be passed to *exclusions* parameter of the `TweakReg` task.

The same thing can be achieved manually using, for example, external FITS viewers, e.g., SAO DS9. For example, based on some image `refimg.fits` we can select a few small regions of sky that contain several good (bright, not saturated) point-like sources that could be used for image alignment of other images (say `img1.fits`, `img2.fits`, etc.). We can save this region file in sky coordinates (e.g., `fk5`), e.g., under the name `input_reg.reg`. We can then load a specific extension of each of the images `img1.fits`, `img2.fits`, etc. one by one into DS9 and then load onto those images the previously created include/exclude region file `input_reg.reg`. Now we can save the regions using *image* coordinates. To do conversion from the sky coordinates to image coordinates, DS9 will use the WCS info from the image onto which the region file was loaded. The *MapReg()* task tries to automate this process.

NOTE 2: *MapReg()* relies on the stregion package for region file parsing and coordinate transformation. Unfortunately, as of writing, **stregion** does not consider distortion corrections when performing coordinate transformations. Therefore, there might be a slight discrepancy between the regions produced by *MapReg()* and the DS9 regions obtained as described in the NOTE 1 above.

NOTE 3: *MapReg()* does not take into account pointing errors and thus the produced region files can be somewhat misaligned compared to their intended position around the sources identified in the "reference" image. Threfore, it is highly recommended that the produced region files be loaded into DS9 and their position be adjusted manually to include the sources of interest (or to avoid the regions that need to be avoided). If possible, the *include* or *exclude* regions should be large enough as to allow for most pointing errors.

Examples

Let's say that based on some image `refimg.fits` we have produced a "master" reference image (`master.reg`) that includes regions around sources that we want to use for image alignment in task *TweakReg()* and excludes regions that we want to avoid being used for image alignment (e.g, diffraction spikes, saturated quasars, stars, etc.). We save the file `master.reg` in sky CS (e.g., `fk5`).

Also, let's assume that we have a set of images `img1.fits`, `img2.fits`, etc. with four FITS extensions named 'SCI' and 'DQ'. For some of the extensions, after analizing the `img*.fits` images we have identified parts of the chips that cannot be used for image alignment. We create region files for those extensions and save the files in image CS as, e.g., `img1_chip_sci2.reg` (in our example this will be the only chip that needs "special" treatment).

Finally, let's say we want to "replicate" the "master" region file to all SCI exctensions of the `img*.fits` images as well as to the 2nd DQ extension and to the 8th extension of the `img*.fits` images.

To do this we run:

```
>>> mapreg(input_reg = 'master.reg', images='img*.fits',
...        img_wcs_ext='sci;dq,2;8', chip_reg='None,
...        img1_chip_sci2.reg,None,None,None,None')
```

This will produce six region files in the ./regions subdirectory for *each* input image:

```
``img1_sci1_twreg.reg``,    ``img1_sci2_twreg.reg``,    ``img1_sci3_twreg.
↪reg``,
``img1_sci4_twreg.reg``,    ``img1_dq2_twreg.reg``,     ``img1_extn8_twreg.
↪reg``
...
```

```
``img2_sci1_twreg.reg``,    ``img2_sci2_twreg.reg``,    ``img2_sci3_twreg.
↪reg``,
``img2_sci4_twreg.reg``,    ``img2_dq2_twreg.reg``,     ``img2_extn8_twreg.
↪reg``
...
```

drizzlepac.mapreg.**map_region_files**(*input_reg*, *images*, *img_wcs_ext='sci'*, *refimg=None*, *ref_wcs_ext='sci'*, *chip_reg=None*, *outpath='./regions'*, *filter=None*, *catfname=None*, *iteractive=False*, *append=False*, *verbose=True*)

drizzlepac.mapreg.**help**(*file=None*)

Print out syntax help for running `mapreg`.

Parameters

file

[str (Default = None)] If given, write out help to the filename specified by this parameter Any previously existing file with this name will be deleted before writing out the help.

TWEAKUTILS: Utility Functions for TWEAKREG

The functions in this module support the various aspects of `TweakReg`, including finding the objects in the images and plotting the residuals.

Authors

Warren Hack

License

License

drizzlepac.tweakutils.**build_pos_grid**(*start*, *end*, *nstep*, *mesh=False*)

Deprecated since version 3.0.0: The build_pos_grid function is deprecated and may be removed in a future version.

Return a grid of positions starting at X,Y given by 'start', and ending at X,Y given by 'end'. The grid will be completely filled in X and Y by every 'step' interval.

drizzlepac.tweakutils.**build_xy_zeropoint**(*imgxy*, *refxy*, *searchrad=3.0*, *histplot=False*, *figure_id=1*, *plotname=None*, *interactive=True*)

Deprecated since version 3.0.0: The build_xy_zeropoint function is deprecated and may be removed in a future version.

Create a matrix which contains the delta between each XY position and
each UV position.

drizzlepac.tweakutils.**createWcsHDU**(*wcs*)

Generate a WCS header object that can be used to populate a reference WCS HDU.

For most applications, stwcs.wcsutil.HSTWCS.wcs2header() will work just as well.

drizzlepac.tweakutils.**find_xy_peak**(*img*, *center=None*, *sigma=3.0*)

Deprecated since version 3.0.0: The find_xy_peak function is deprecated and may be removed in a future version.

Find the center of the peak of offsets

drizzlepac.tweakutils.**gauss**(*x*, *sigma*)

Compute 1-D value of gaussian at position x relative to center.

drizzlepac.tweakutils.**gauss_array**(*nx*, *ny=None*, *fwhm=1.0*, *sigma_x=None*, *sigma_y=None*, *zero_norm=False*)

Computes the 2D Gaussian with size nx*ny.

Parameters

nx
[int]

ny
[int [Default: None]] Size of output array for the generated Gaussian. If ny == None, output will be an array nx X nx pixels.

fwhm
[float [Default: 1.0]] Full-width, half-maximum of the Gaussian to be generated

sigma_x
[float [Default: None]]

sigma_y
[float [Default: None]] Sigma_x and sigma_y are the stddev of the Gaussian functions.

zero_norm
[bool [Default: False]] The kernel will be normalized to a sum of 1 when True.

Returns

gauss_arr
[array] A numpy array with the generated gaussian function

drizzlepac.tweakutils.**idlgauss_convolve**(*image*, *fwhm*)

Deprecated since version 3.0.0: The idlgauss_convolve function is deprecated and may be removed in a future version.

drizzlepac.tweakutils.**isfloat**(*value*)

Return True if all characters are part of a floating point value

drizzlepac.tweakutils.**make_vector_plot**(*coordfile*, *columns=[1, 2, 3, 4]*, *data=None*, *figure_id=None*, *title=None*, *axes=None*, *every=1*, *labelsize=8*, *ylimit=None*, *limit=None*, *xlower=None*, *ylower=None*, *output=None*, *headl=4*, *headw=3*, *xsh=0.0*, *ysh=0.0*, *fit=None*, *scale=1.0*, *vector=True*, *textscale=5*, *append=False*, *linfit=False*, *rms=True*, *plotname=None*)

Convert a XYXYMATCH file into a vector plot or set of residuals plots.

This function provides a single interface for generating either a vector plot of residuals or a set of 4 plots showing residuals. The data being plotted can also be adjusted for a linear fit on-the-fly.

Parameters

coordfile
[string] Name of file with matched sets of coordinates. This input file can be a file compatible for use with IRAF's geomap.

columns
[list [Default: [0,1,2,3]]] Column numbers for the X,Y positions from each image

data
[list of arrays] If specified, this can be used to input matched data directly

title
[string] Title to be used for the generated plot

axes
[list] List of X and Y min/max values to customize the plot axes

every
[int [Default: 1]] Slice value for the data to be plotted

limit
[float] Radial offset limit for selecting which sources are included in the plot

labelsize
[int [Default: 8] or str] Font size to use for tick labels, either in font points or as a string understood by tick_params().

ylimit
[float] Limit to use for Y range of plots.

xlower
[float]

ylower
[float] Limit in X and/or Y offset for selecting which sources are included in the plot

output
[string] Filename of output file for generated plot

headl
[int [Default: 4]] Length of arrow head to be used in vector plot

headw
[int [Default: 3]] Width of arrow head to be used in vector plot

xsh
[float]

ysh
[float] Shift in X and Y from linear fit to be applied to source positions from the first image

scale
[float] Scale from linear fit to be applied to source positions from the first image

fit
[array] Array of linear coefficients for rotation (and scale?) in X and Y from a linear fit to be applied to source positions from the first image

vector
[bool [Default: True]] Specifies whether or not to generate a vector plot. If False, task will generate a set of 4 residuals plots instead

textscale
[int [Default: 5]] Scale factor for text used for labelling the generated plot

append
[bool [Default: False]] If True, will overplot new plot on any pre-existing plot

linfit
[bool [Default: False]] If True, a linear fit to the residuals will be generated and added to the generated residuals plots

rms
[bool [Default: True]] Specifies whether or not to report the RMS of the residuals as a label on the generated plot(s).

plotname
[str [Default: None]] Write out plot to a file with this name if specified.

drizzlepac.tweakutils.**parse_atfile_cat**(*input*)

Return the list of catalog filenames specified as part of the input @-file

drizzlepac.tweakutils.**parse_colname**(*colname*)

Common function to interpret input column names provided by the user.

This function translates column specification provided by the user into a column number.

Parameters

colname
Column name or names to be interpreted

Returns

cols
[list] The return value will be a list of strings.

Notes

This function will understand the following inputs:

```
'1,2,3' or   'c1,c2,c3' or ['c1','c2','c3']
'1-3'   or   'c1-c3'
'1:3'   or   'c1:c3'
'1 2 3' or   'c1 c2 c3'
'1'     or   'c1'
1
```

drizzlepac.tweakutils.**parse_exclusions**(*exclusions*)

Read in exclusion definitions from file named by 'exclusions' and return a list of positions and distances

drizzlepac.tweakutils.**parse_skypos**(*ra*, *dec*)

Function to parse RA and Dec input values and turn them into decimal degrees

Input formats could be:
["nn","nn","nn.nn"] "nn nn nn.nnn" "nn:nn:nn.nn" "nnH nnM nn.nnS" or "nnD nnM nn.nnS" nn.nnnnnnnn "nn.nnnnnnn"

drizzlepac.tweakutils.**plot_zeropoint**(*pars*)

Plot 2d histogram.

Pars will be a dictionary containing:
data, figure_id, vmax, title_str, xp,yp, searchrad

drizzlepac.tweakutils.**radec_hmstodd**(*ra*, *dec*)

Function to convert HMS values into decimal degrees.

This function relies on the astropy.coordinates package to perform the conversion to decimal degrees.

Parameters

ra
[list or array] List or array of input RA positions

dec
[list or array] List or array of input Dec positions

Returns

pos
[arr] Array of RA,Dec positions in decimal degrees

See also:

`astropy.coordinates`

Notes

This function supports any specification of RA and Dec as HMS or DMS; specifically, the formats:

```
["nn","nn","nn.nn"]
"nn nn nn.nnn"
"nn:nn:nn.nn"
"nnH nnM nn.nnS" or "nnD nnM nn.nnS"
```

drizzlepac.tweakutils.**read_ASCII_cols**(*infile*, *cols=[1, 2, 3]*)

Interpret input ASCII file to return arrays for specified columns.

Returns

outarr
[list of arrays] The return value will be a list of numpy arrays, one for each 'column'.

Notes

The specification of the columns should be expected to have lists for each 'column', with all columns in each list combined into a single entry.

For example:

```
cols = ['1,2,3','4,5,6',7]
```

where '1,2,3' represent the X/RA values, '4,5,6' represent the Y/Dec values and 7 represents the flux value for a total of 3 requested columns of data to be returned.

drizzlepac.tweakutils.**read_FITS_cols**(*infile*, *cols=None*)

Read columns from FITS table

drizzlepac.tweakutils.**readcols**(*infile*, *cols=None*)

Function which reads specified columns from either FITS tables or ASCII files

This function reads in the columns specified by the user into numpy arrays regardless of the format of the input table (ASCII or FITS table).

Parameters

infile
[string] Filename of the input file

cols
[string or list of strings] Columns to be read into arrays

Returns

outarr
[array] Numpy array or arrays of columns from the table

drizzlepac.tweakutils.**write_shiftfile**(*image_list*, *filename*, *outwcs='tweak_wcs.fits'*)

Write out a shiftfile for a given list of input Image class objects

TWEAKBACK: Propagating The Solution

`tweakback` - propagate the "tweaked" solutions back to the original input files.

Version 0.4.0 - replaced previous algorithm that used fitting of WCS footprints to reconstruct the transformation that was applied to the old drizzled image (to align it with another image) to obtain the new drizzled image WCS with an algorithm that is based on linearization of the exact compound operator that transforms current image coordinates to the "aligned" (to the new drizzled WCS) image coordinates.

Authors
Warren Hack, Mihai Cara

License
License

drizzlepac.tweakback.**apply_tweak**(*drz_file*, *orig_wcs_name*, *output_wcs_name=None*, *input_files=None*, *default_extname='SCI'*, ***kwargs*)

Apply WCS solution recorded in drizzled file to distorted input images (`_flt.fits` files) used to create the drizzled file.

It is assumed that if input images given by `input_files` are drizzled together, they would produce the drizzled image given by `drz_file` image and with the "original" primary WCS. It is also assumed that drizzled image was aligned using `tweakreg` either to another image or to an external reference catalog. We will refer to the primary WCS in the drizzled image **before** `tweakreg` was run as the "original" WCS and the WCS **after** `tweakreg` was run as "tweaked" WCS.

By comparing both "original" and "tweaked" WCS, `apply_wcs` computes the correction that was applied by `tweakreg` to the "original" WCS and converts this correction in the drizzled image frame into a correction in the input image's (`input_files`) frame that will be applied to the primary WCS of input images. If updated input images are now resampled again, they would produce an image

very close to drz_file but with a primary WCS very similar to the "tweaked" WCS instead of the "original" WCS.

Parameters

drz_file
[str] File name of the drizzled image that contains both the "original" and "tweaked" WCS. Even though wildcards are allowed in the file name, their expansion must resolve to a single image file. By default, apply_tweak looks for the first image-like HDU in the drizzled image. To specify a particular extension from which to load WCS, append extension specification after the file name, for example:

- 'image_drz.fits[sci,1]' for first "sci" extension
- 'image_drz.fits[1]' for the first extension
- 'image_drz.fits[0]' for the primary HDU

orig_wcs_name
[str] Name (provided by the WCSNAME? header keyword where ? respesents a letter A-Z) of the "original" WCS. This is the WCS of the resampled image (obtained by drizzling all input images) **before** this resampled image was aligned ("tweaked") to another image/catalog.

If orig_wcs_name is None, the the original WCS **must be specified** using orig_wcs_key. When orig_wcs_key is provided, orig_wcs_name is ignored altogether.

output_wcs_name
[str, None] Value of WCSNAME to be used to label the updated solution in the input (e.g., _flt.fits) files. If left blank or None, it will default to using either the current (primary) WCSNAME value from the drz_file or from the alternate WCS given by the tweaked_wcs_name or tweaked_wcs_key parameters.

input_files
[str, None] Filenames of distorted images whose primary WCS is to be updated with the same transformation as used in the "tweaked" drizzled image. Default value of None indicates that input image filenames will be derived from the D*DATA keywords written out by the AstroDrizzle. If they can not be found, the task will quit.

input_files string can contain one of the following:

- a comma-separated list of valid science image file names (see note below) and (optionally) extension specifications, e.g.: 'j1234567q_flt.fits[1], j1234568q_flt.fits[sci,2]';
- an @-file name, e.g., '@files_to_match.txt'.

Note: Valid science image file names are:

- file names of existing FITS, GEIS, or WAIVER FITS files;

- partial file names containing wildcard characters, e.g., `'*_flt.fits'`;
- Association (ASN) tables (must have `_asn`, or `_asc` suffix), e.g., `'j12345670_asn.fits'`.

Warning: @-file names **MAY NOT** be followed by an extension specification.

Warning: If an association table or a partial file name with wildcard characters is followed by an extension specification, it will be considered that this extension specification applies to **each** file name in the association table or **each** file name obtained after wildcard expansion of the partial file name.

default_extname
[str] Extension name of extensions in input images whose primary WCS should be updated. This value is used only when file names provided in `input_files` do not contain extension specifications.

Other Parameters

tweaked_wcs_name
[str] Name of the "tweaked" WCS. This is the WCS of the resampled image (obtained by drizzling all input images) _after_ this resampled image was aligned ("tweaked") to another image/catalog.

When neither `tweaked_wcs_name` nor ``tweaked_wcs_key` are not provided, `apply_tweak` will take the current primary WCS in the drizzled image as a "tweaked" WCS. `tweaked_wcs_name` is ignored when `tweaked_wcs_key` is provided.

tweaked_wcs_key
[{' ', 'A'-'Z'}] Same as `tweaked_wcs_name` except it specifies a WCS by key instead of name. When provided, `tweaked_wcs_name` is ignored.

orig_wcs_key
[{' ', 'A'-'Z'}] Same as `orig_wcs_name` except it specifies a WCS by key instead of name. When provided, `orig_wcs_name` is ignored.

See also:

`stwcs.wcsutil.altwcs`

Notes

The algorithm used by this function is based on linearization of the exact compound operator that converts input image coordinates to the coordinates (in the input image) that would result in alignment with the new drizzled image WCS.

Warning: Parameters `orig_wcs_name` and `tweaked_wcs_name` (or their "key" equivalents) allow computation of transformation between *any two WCS* in the drizzled image and application of this transformation to the primary WCS of the input images. This will produce an expected result **only if** the WCS pointed to by `orig_wcs_name` was obtained by drizzling input images with their current primary WCS.

Examples

A drizzled image named `acswfc_mos2_drz.fits` was created from 4 images using `AstroDrizzle`. The primary WCS of this drizzled image was named `'INITIAL_GUESS'`. This drizzled image was then aligned to some other image using `TweakReg` and the updated ("tweaked") primary WCS was named `'BEST_WCS'` while the previous primary WCS - the WCS named `'INITIAL_GUESS'` - was archived by `TweakReg` under WCS key `'C'`. We will refer to this archived WCS as the "original" WCS. `apply_tweak` can now be used to compute the transformation between the original and the tweaked WCS and apply this transformation to the WCS of each of the input images that were drizzle-combined to produce the resampled image `acswfc_mos2_drz.fits`.

The simplest way to accomplish this would be to run `apply_tweak()` using default parameters:

```
>>> from drizzlepac import tweakback
>>> tweakback.apply_tweak('acswfc_mos2_drz.fits', orig_wcs_name='INITIAL_
↪GUESS')
```

or

```
>>> tweakback.apply_tweak('acswfc_mos2_drz.fits', orig_wcs_key='C')
```

If the same WCS should be applied to a specific set of images or extensions in those images, then we can explicitly specify input files:

```
>>> tweakback.apply_tweak(
...     'acswfc_mos2_drz.fits',
...     input='img_mos2a_flt.fits,img_mos2c_flt.fits[1],img_mos2d_flt.
↪fits[sci,1]'
... )
```

In the examples above, current primary WCS of the input `'img_mos2?_flt.fits'` files will be archived and the primary WCS will be replaced with a "tweaked" WCS obtained by applying relevant transformations to the current primary WCS. Because we did not specify `output_wcs_name`, the name of this tweaked primary WCS in the input images will be set to `'BEST_WCS'`.

drizzlepac.tweakback.**determine_extnum**(*drzfile*, *extname='SCI'*)

drizzlepac.tweakback.**determine_orig_wcsname**(*header*, *wnames*, *wkeys*)

Determine the name of the original, unmodified WCS solution

drizzlepac.tweakback.**extract_input_filenames**(*drzfile*)

Generate a list of filenames from a drizzled image's header

drizzlepac.tweakback.**linearize**(*wcsim*, *wcsima*, *wcs_olddrz*, *wcs_newdrz*, *imcrpix*, *hx=1.0*, *hy=1.0*)

drizzlepac.tweakback.**run**(*configobj*)

drizzlepac.tweakback.**update_chip_wcs**(*chip_wcs*, *drz_old_wcs*, *drz_new_wcs*, *xrms=None*, *yrms=None*)

drizzlepac.tweakback.**help**(*file=None*)

Print out syntax help for running `tweakback`.

Parameters

file

[str (Default = None)] If given, write out help to the filename specified by this parameter Any previously existing file with this name will be deleted before writing out the help.

drizzlepac.tweakback.**apply_tweak**(*drz_file*, *orig_wcs_name*, *output_wcs_name=None*, *input_files=None*, *default_extname='SCI'*, ***kwargs*)

Apply WCS solution recorded in drizzled file to distorted input images (`_flt.fits` files) used to create the drizzled file.

It is assumed that if input images given by `input_files` are drizzled together, they would produce the drizzled image given by `drz_file` image and with the "original" primary WCS. It is also assumed that drizzled image was aligned using `tweakreg` either to another image or to an external reference catalog. We will refer to the primary WCS in the drizzled image **before** `tweakreg` was run as the "original" WCS and the WCS **after** `tweakreg` was run as "tweaked" WCS.

By comparing both "original" and "tweaked" WCS, `apply_wcs` computes the correction that was applied by `tweakreg` to the "original" WCS and converts this correction in the drizzled image frame into a correction in the input image's (`input_files`) frame that will be applied to the primary WCS of input images. If updated input images are now resampled again, they would produce an image very close to `drz_file` but with a primary WCS very similar to the "tweaked" WCS instead of the "original" WCS.

Parameters

drz_file

[str] File name of the drizzled image that contains both the "original" and "tweaked" WCS. Even though wildcards are allowed in the file name, their expansion must resolve to a single image file. By default, `apply_tweak` looks for the first image-like HDU in the drizzled image. To specify a particular extension from which to load WCS, append extension specification after the file name, for example:

- `'image_drz.fits[sci,1]'` for first "sci" extension
- `'image_drz.fits[1]'` for the first extension
- `'image_drz.fits[0]'` for the primary HDU

orig_wcs_name

[str] Name (provided by the `WCSNAME?` header keyword where `?` respesents a letter A-Z) of the "original" WCS. This is the WCS of the resampled image (obtained by drizzling all input images) **before** this resampled image was aligned ("tweaked") to another image/catalog.

If `orig_wcs_name` is `None`, the the original WCS **must be specified** using `orig_wcs_key`. When `orig_wcs_key` is provided, `orig_wcs_name` is ignored altogether.

output_wcs_name

[str, None] Value of `WCSNAME` to be used to label the updated solution in the input (e.g., `_flt.fits`) files. If left blank or `None`, it will default to using either the current (primary) `WCSNAME` value from the `drz_file` or from the alternate WCS given by the `tweaked_wcs_name` or `tweaked_wcs_key` parameters.

input_files

[str, None] Filenames of distorted images whose primary WCS is to be updated with the same transformation as used in the "tweaked" drizzled image. Default value of `None` indicates that input image filenames will be derived from the `D*DATA` keywords written out by the `AstroDrizzle`. If they can not be found, the task will quit.

`input_files` string can contain one of the following:

- a comma-separated list of valid science image file names (see note below) and (optionally) extension specifications, e.g.: `'j1234567q_flt.fits[1], j1234568q_flt.fits[sci,2]'`;
- an @-file name, e.g., `'@files_to_match.txt'`.

Note: Valid science image file names are:

- file names of existing FITS, GEIS, or WAIVER FITS files;
- partial file names containing wildcard characters, e.g., `'*_flt.fits'`;
- Association (ASN) tables (must have `_asn`, or `_asc` suffix), e.g., `'j12345670_asn.fits'`.

Warning: @-file names **MAY NOT** be followed by an extension specification.

> **Warning:** If an association table or a partial file name with wildcard characters is followed by an extension specification, it will be considered that this extension specification applies to **each** file name in the association table or **each** file name obtained after wildcard expansion of the partial file name.

default_extname
: [str] Extension name of extensions in input images whose primary WCS should be updated. This value is used only when file names provided in `input_files` do not contain extension specifications.

Other Parameters

tweaked_wcs_name
: [str] Name of the "tweaked" WCS. This is the WCS of the resampled image (obtained by drizzling all input images) _after_ this resampled image was aligned ("tweaked") to another image/catalog.

  When neither `tweaked_wcs_name` nor ``tweaked_wcs_key` are not provided, `apply_tweak` will take the current primary WCS in the drizzled image as a "tweaked" WCS. `tweaked_wcs_name` is ignored when `tweaked_wcs_key` is provided.

tweaked_wcs_key
: [{' ', 'A'-'Z'}] Same as `tweaked_wcs_name` except it specifies a WCS by key instead of name. When provided, `tweaked_wcs_name` is ignored.

orig_wcs_key
: [{' ', 'A'-'Z'}] Same as `orig_wcs_name` except it specifies a WCS by key instead of name. When provided, `orig_wcs_name` is ignored.

See also:

`stwcs.wcsutil.altwcs`

Notes

The algorithm used by this function is based on linearization of the exact compound operator that converts input image coordinates to the coordinates (in the input image) that would result in alignment with the new drizzled image WCS.

> **Warning:** Parameters `orig_wcs_name` and `tweaked_wcs_name` (or their "key" equivalents) allow computation of transformation between *any two WCS* in the drizzled image and application of this transformation to the primary WCS of the input images. This will produce an expected result **only if** the WCS pointed to by `orig_wcs_name` was obtained by drizzling input images with their current primary WCS.

Examples

A drizzled image named `acswfc_mos2_drz.fits` was created from 4 images using `AstroDrizzle`. The primary WCS of this drizzled image was named `'INITIAL_GUESS'`. This drizzled image was then aligned to some other image using `TweakReg` and the updated ("tweaked") primary WCS was named `'BEST_WCS'` while the previous primary WCS - the WCS named `'INITIAL_GUESS'` - was archived by `TweakReg` under WCS key `'C'`. We will refer to this archived WCS as the "original" WCS. `apply_tweak` can now be used to compute the transformation between the original and the tweaked WCS and apply this transformation to the WCS of each of the input images that were drizzle-combined to produce the resampled image `acswfc_mos2_drz.fits`.

The simplest way to accomplish this would be to run `apply_tweak()` using default parameters:

```
>>> from drizzlepac import tweakback
>>> tweakback.apply_tweak('acswfc_mos2_drz.fits', orig_wcs_name='INITIAL_
↪GUESS')
```

or

```
>>> tweakback.apply_tweak('acswfc_mos2_drz.fits', orig_wcs_key='C')
```

If the same WCS should be applied to a specific set of images or extensions in those images, then we can explicitly specify input files:

```
>>> tweakback.apply_tweak(
...     'acswfc_mos2_drz.fits',
...     input='img_mos2a_flt.fits,img_mos2c_flt.fits[1],img_mos2d_flt.
↪fits[sci,1]'
... )
```

In the examples above, current primary WCS of the input `'img_mos2?_flt.fits'` files will be archived and the primary WCS will be replaced with a "tweaked" WCS obtained by applying relevant transformations to the current primary WCS. Because we did not specify `output_wcs_name`, the name of this tweaked primary WCS in the input images will be set to `'BEST_WCS'`.

UPDATEHDR: Functions for Updating WCS with New Solutions

The functions in this module support updating the WCS information in distorted images with the alignment solution determined by `TweakReg` or saved in a shiftfile.

Authors

Warren Hack, Mihai Cara

License

License

`drizzlepac.updatehdr.`**`create_unique_wcsname`**(*fimg*, *extnum*, *wcsname*)

This function evaluates whether the specified wcsname value has already been used in this image. If so, it automatically modifies the name with a simple version ID using wcsname_NNN format.

Parameters

fimg
[obj] PyFITS object of image with WCS information to be updated

extnum
[int] Index of extension with WCS information to be updated

wcsname
[str] Value of WCSNAME specified by user for labelling the new WCS

Returns

uniqname
[str] Unique WCSNAME value

drizzlepac.updatehdr.**interpret_wcsname_type**(*wcsname*)

Interpret WCSNAME as a standardized human-understandable description

drizzlepac.updatehdr.**linearize**(*wcsim*, *wcsima*, *wcsref*, *imcrpix*, *f*, *shift*, *hx=1.0*, *hy=1.0*)

linearization using 5-point formula for first order derivative

drizzlepac.updatehdr.**update_from_shiftfile**(*shiftfile*, *wcsname=None*, *force=False*)

Update headers of all images specified in shiftfile with shifts from shiftfile.

Parameters

shiftfile
[str] Filename of shiftfile.

wcsname
[str] Label to give to new WCS solution being created by this fit. If a value of None is given, it will automatically use 'TWEAK' as the label. [Default =None]

force
[bool] Update header even though WCS already exists with this solution or wcsname? [Default=False]

drizzlepac.updatehdr.**update_refchip_with_shift**(*chip_wcs*, *wcslin*, *fitgeom='rscale'*, *rot=0.0*, *scale=1.0*, *xsh=0.0*, *ysh=0.0*, *fit=None*, *xrms=None*, *yrms=None*)

Compute the matrix for the scale and rotation correction

Parameters

chip_wcs: wcs object
HST of the input image

wcslin: wcs object
Reference WCS from which the offsets/rotations are determined

fitgeom: str
NOT USED

rot
[float] Amount of rotation measured in fit to be applied. [Default=0.0]

scale
[float] Amount of scale change measured in fit to be applied. [Default=1.0]

xsh
[float] Offset in X pixels from defined tangent plane to be applied to image. [Default=0.0]

ysh
[float] Offset in Y pixels from defined tangent plane to be applied to image. [Default=0.0]

fit
[arr] Linear coefficients for fit [Default = None]

xrms
[float] RMS of fit in RA (in decimal degrees) that will be recorded as CRDER1 in WCS and header [Default = None]

yrms
[float] RMS of fit in Dec (in decimal degrees) that will be recorded as CRDER2 in WCS and header [Default = None]

drizzlepac.updatehdr.**update_wcs**(*image*, *extnum*, *new_wcs*, *wcsname=''*, *reusename=False*, *verbose=False*)

Updates the WCS of the specified extension number with the new WCS after archiving the original WCS.

The value of 'new_wcs' needs to be the full HSTWCS object.

Parameters

image
[str] Filename of image with WCS that needs to be updated

extnum
[int] Extension number for extension with WCS to be updated/replaced

new_wcs
[object] Full HSTWCS object which will replace/update the existing WCS

wcsname
[str] Label to give newly updated WCS

reusename
[bool] User can choose whether to over-write WCS with same name or not. [Default: False]

verbose
[bool, int] Print extra messages during processing? [Default: False]

drizzlepac.updatehdr.**updatewcs_with_shift**(*image*, *reference*, *wcsname='TWEAK'*, *reusename=False*, *fitgeom='rscale'*, *rot=0.0*, *scale=1.0*, *xsh=0.0*, *ysh=0.0*, *fit=None*, *xrms=None*, *yrms=None*, *verbose=False*, *force=False*, *sciext='SCI'*)

Update the SCI headers in 'image' based on the fit provided as determined in the WCS specified by 'reference'. The fit should be a 2-D matrix as generated for use with 'make_vector_plot()'.

Parameters

image
[str or PyFITS.HDUList object] Filename, or PyFITS object, of image with WCS to be updated. All extensions with EXTNAME matches the value of the 'sciext' parameter value (by default, all 'SCI' extensions) will be updated.

reference
[str] Filename of image/headerlet (FITS file) which contains the WCS used to define the tangent plane in which all the fit parameters (shift, rot, scale) were measured.

wcsname
[str, None, optional] Label to give to new WCS solution being created by this fit. If a value of None is given, it will automatically use 'TWEAK' as the label. [Default =None]

reusename
[bool] User can specify whether or not to over-write WCS with same name. [Default: False]

rot
[float] Amount of rotation measured in fit to be applied. [Default=0.0]

scale
[float] Amount of scale change measured in fit to be applied. [Default=1.0]

xsh
[float] Offset in X pixels from defined tangent plane to be applied to image. [Default=0.0]

ysh
[float] Offset in Y pixels from defined tangent plane to be applied to image. [Default=0.0]

fit
[arr] Linear coefficients for fit [Default = None]

xrms
[float] RMS of fit in RA (in decimal degrees) that will be recorded as CRDER1 in WCS and header [Default = None]

yrms
[float] RMS of fit in Dec (in decimal degrees) that will be recorded as CRDER2 in WCS and header [Default = None]

verbose
[bool] Print extra messages during processing? [Default=False]

force

[bool] Update header even though WCS already exists with this solution or wcsname? [Default=False]

sciext
[string] Value of FITS EXTNAME keyword for extensions with WCS headers to be updated with the fit values. [Default='SCI']

Notes

The algorithm used to apply the provided fit solution to the image involves applying the following steps to the WCS of each of the input image's chips:

1. **compute RA/Dec with full distortion correction for**
   reference point as (Rc_i,Dc_i)
2. **find the Xc,Yc for each Rc_i,Dc_i and get the difference from the**
   CRPIX position for the reference WCS as (dXc_i,dYc_i)
3. **apply fit (rot&scale) to (dXc_i,dYc_i) then apply shift, then add**
   CRPIX back to get new (Xcs_i,Ycs_i) position
4. compute (Rcs_i,Dcs_i) as the sky coordinates for (Xcs_i,Ycs_i)
5. compute delta of (Rcs_i-Rc_i, Dcs_i-Dcs_i) as (dRcs_i,dDcs_i)
6. **apply the fit to the chip's undistorted CD matrix, the apply linear**
   distortion terms back in to create a new CD matrix
7. add (dRcs_i,dDcs_i) to CRVAL of the reference chip's WCS
8. update header with new WCS values

Functions to Manage WCS Table Extension

These functions provide the basic support for initializing, creating and updating the WCS table extension which serves as the archive of updates made to the WCS information in the image headers.

stwcs.wcsutil.wcscorr.**archive_wcs_file**(*image*, *wcs_id=None*)

Update WCSCORR table with rows for each SCI extension to record the newly updated WCS keyword values.

stwcs.wcsutil.wcscorr.**create_wcscorr**(*descrip=False*, *numrows=1*, *padding=0*)

Return the basic definitions for a WCSCORR table. The dtype definitions for the string columns are set to the maximum allowed so that all new elements will have the same max size which will be automatically truncated to this limit upon updating (if needed).

The table is initialized with rows corresponding to the OPUS solution for all the 'SCI' extensions.

stwcs.wcsutil.wcscorr.**delete_wcscorr_row**(*wcstab*, *selections=None*, *rows=None*)

Sets all values in a specified row or set of rows to default values

This function will essentially erase the specified row from the table without actually removing the row from the table. This avoids the problems with trying to resize the number of rows in the table while preserving the ability to update the table with new rows again without resizing the table.

Parameters

wcstab: object
PyFITS binTable object for WCSCORR table

selections: dict
Dictionary of wcscorr column names and values to be used to select the row or set of rows to erase

rows: int, list
If specified, will specify what rows from the table to erase regardless of the value of 'selections'

stwcs.wcsutil.wcscorr.**find_wcscorr_row**(*wcstab*, *selections*)

Return an array of indices from the table (NOT HDU) 'wcstab' that matches the selections specified by the user.

The row selection criteria must be specified as a dictionary with column name as key and value(s) representing the valid desired row values. For example, {'wcs_id':'OPUS','extver':2}.

stwcs.wcsutil.wcscorr.**init_wcscorr**(*input*, *force=False*)

This function will initialize the WCSCORR table if it is not already present, and look for WCS keywords with a prefix of 'O' as the original OPUS generated WCS as the initial row for the table or use the current WCS keywords as initial row if no 'O' prefix keywords are found.

This function will NOT overwrite any rows already present.

This function works on all SCI extensions at one time.

stwcs.wcsutil.wcscorr.**restore_file_from_wcscorr**(*image*, *id='OPUS'*, *wcskey=''*)

Copies the values of the WCS from the WCSCORR based on ID specified by user. The default will be to restore the original OPUS-derived values to the Primary WCS. If wcskey is specified, the WCS with that key will be updated instead.

stwcs.wcsutil.wcscorr.**update_wcscorr**(*dest*, *source=None*, *extname='SCI'*, *wcs_id=None*, *active=True*)

Update WCSCORR table with a new row or rows for this extension header. It copies the current set of WCS keywords as a new row of the table based on keyed WCSs as per Paper I Multiple WCS standard).

Parameters

dest
[HDUList] The HDU list whose WCSCORR table should be appended to (the WCSCORR HDU must already exist)

source
[HDUList, optional] The HDU list containing the extension from which to extract the WCS keywords to add to the WCSCORR table. If None, the dest is also used as the source.

extname
[str, optional] The extension name from which to take new WCS keywords. If there are multiple extensions with that name, rows are added for each extension version.

wcs_id
[str, optional] The name of the WCS to add, as in the WCSNAMEa keyword. If unspecified, all the WCSs in the specified extensions are added.

active: bool, optional
When True, indicates that the update should reflect an update of the active WCS information, not just appending the WCS to the file as a headerlet

stwcs.wcsutil.wcscorr.**update_wcscorr_column**(*wcstab*, *column*, *values*, *selections=None*, *rows=None*)

Update the values in 'column' with 'values' for selected rows

Parameters

wcstab: object
PyFITS binTable object for WCSCORR table

column: string
Name of table column with values that need to be updated

values: string, int, or list
Value or set of values to copy into the selected rows for the column

selections: dict
Dictionary of wcscorr column names and values to be used to select the row or set of rows to erase

rows: int, list
If specified, will specify what rows from the table to erase regardless of the value of 'selections'

Functions to Manage Legacy OPUS WCS Keywords in the WCS Table

The previously released versions of `makewcs` provided with `MultiDrizzle` *archives* the original OPUS generated WCS keywords using header keywords which have a prefix of "O", such as "OCRPIX1". In order to avoid overwriting or ignoring these original values, these functions can be used to convert the prefixed OPUS WCS keywords into WCS table entries compatible with the new code.

Strictly to provide complete support for these OPUS keywords, the code will also create, if the user desires, prefix "O" WCS keywords from the alternate WCS FITS conventions OPUS keywords. This would allow images processed using the new code only can then be used with older versions of `MultiDrizzle`, if the user needs such compatibility.

stwcs.wcsutil.convertwcs.**archive_prefix_OPUS_WCS**(*fobj*, *extname='SCI'*)

Identifies WCS keywords which were generated by OPUS and archived using a prefix of 'O' for all 'SCI' extensions in the file

Parameters

fobj
[str or astropy.io.fits.HDUList] Filename or fits object of a file

stwcs.wcsutil.convertwcs.**create_prefix_OPUS_WCS**(*fobj*, *extname='SCI'*)

Creates alternate WCS with a prefix of 'O' for OPUS generated WCS values to work with old MultiDrizzle.

Parameters

fobj
[str or astropy.io.fits.HDUList] Filename or fits object of a file

Raises

IOError:
if input FITS object was not opened in 'update' mode

CHAPTER

FIVE

UTILITIES

5.1 Coordinate Transformation Tools

5.1.1 pixtopix: Coordinate transformation to/from drizzled images

This task allows a user to perform coordinate transformations with the full WCS and distortion model to and from drizzled image positions. This task serves as a replacement for the STSDAS.dither task 'tran'. `pixtopix` transforms pixel positions given as X,Y values into positions in another WCS frame based on the WCS information and any recognized distortion keywords from the input image header.

Authors
Warren Hack

License
License

Parameters

inimage
[str] full filename with path of input image, an extension name `['sci',1]` should be provided if input is a multi-extension `FITS` file

outimage
[str, optional] full filename with path of output image, an extension name `['sci',1]` should be provided if output is a multi-extension `FITS` file. If no image gets specified, the input image will be used to generate a default output WCS using `stwcs.distortion.util.output_wcs()`.

direction
[str] Direction of transform (forward or backward). The 'forward' transform takes the pixel positions (assumed to be from the 'input' image) and determines their position in the 'output' image. The 'backward' transform converts the pixel positions (assumed to be from the 'output' image) into pixel positions in the 'input' image.

Optional Parameters

x
: [float, optional] X position from image

y
: [float, optional] Y position from image

coords
: [str, deprecated] [DEPRECATED] full filename with path of file with x,y coordinates Filename given here will be *ignored* if a file has been specified in `coordfile` parameter.

coordfile
: [str, optional] full filename with path of file with starting x,y coordinates

colnames
: [str, optional] comma separated list of column names from 'coords' files containing x,y coordinates, respectively. Will default to first two columns if None are specified. Column names for ASCII files will use 'c1','c2',… convention.

separator
: [str, optional] non-blank separator used as the column delimiter in the coords file

precision
: [int, optional] Number of floating-point digits in output values

output
: [str, optional] Name of output file with results, if desired

verbose
: [bool] Print out full list of transformation results (default: False)

RETURNS

outx
: [float] X position of transformed pixel. If more than 1 input value, then it will be a numpy array.

outy
: [float] Y position of transformed pixel. If more than 1 input value, then it will be a numpy array.

Notes

This task performs a full distortion-correction coordinate transformation based on all WCS keywords and any recognized distortion keywords from the input image header. The transformation recognizes the conventions for describing distortion implemented as part of the SIP and Paper IV conventions used with AstroDrizzle. Input images can be updated to use these conventions through the use of the 'updatewcs' module the STWCS package.

See Also

stwcs

Examples

These examples illustrate the syntax that can be used to run the task in a couple of common modes.

1. Convert the position 256,256 from 'input_flt.fits[sci,1]' into a position on the output image 'output_drz.fits[sci,1]' using:

```
>>> from drizzlepac import pixtopix
>>> outx,outy = pixtopix.tran("input_file_flt.fits[sci,1]",
...                           "output_drz.fits[sci,1],"forward", 256,256)
```

   or if you the default direction of 'forward' is already appropriate and you don't want to explicitly set that value:

```
>>> outx,outy = pixtopix.tran("input_file_flt.fits[sci,1]",
...                           "output_drz.fits[sci,1],x=256,y=256)
```

2. The set of X,Y positions from 'output_drz.fits[sci,1]' stored as the 3rd and 4th columns from the ASCII file `xy_sci1.dat` will be transformed into pixel positions from 'input_flt.fits[sci,1]' and written out to `xy_flt1.dat` using:

```
>>> from drizzlepac import pixtopix
>>> x,y = pixtopix.tran("input_flt.fits[sci,1]",
...          "output_drz.fits[sci,1]", "backward",
...          coordfile='xy_sci1.dat', colnames=['c3','c4'],
...          output="xy_flt1.dat")
```

drizzlepac.pixtopix.**tran**(*inimage*, *outimage*, *direction='forward'*, *x=None*, *y=None*, *coordfile=None*, *colnames=None*, *separator=None*, *precision=6*, *output=None*, *verbose=True*)

Primary interface to perform coordinate transformations in pixel coordinates between 2 images using STWCS and full distortion models read from each image's header.

drizzlepac.pixtopix.**help**(*file=None*)

Print out syntax help for running `pixtopix`.

Parameters

file

[str (Default = None)] If given, write out help to the filename specified by this parameter Any previously existing file with this name will be deleted before writing out the help.

5.1.2 pixtosky: Coordinate transformation to sky coordinates

This task allows a user to perform coordinate transformations with the full WCS and distortion model on source positions from an input image to sky coordinates. This task serves as a replacement for the IRAF.STSDAS task `xy2rd`, albeit with the added capability of understanding the full distortion model provided in the headers of images updated for use with astrodrizzle and tweakreg. `pixtosky` transforms pixel positions given as X,Y values into positions on the sky based on the WCS information and any recognized distortion keywords from the input image header.

Authors
Warren Hack

License
License

Parameters

input
[str] full filename with path of input image, an extension name ['sci',1] should be provided if input is a multi-extension FITS file.

Optional Parameters

x
[float, optional] X position from input image

y
[float, optional] Y position from input image

coordfile
[str, optional] full filename with path of file with x, y coordinates

colnames
[str, optional] comma separated list of column names from 'coordfile' file containing x, y coordinates. Will default to first two columns if None are specified. Column names for ASCII files will use 'c1','c2',... convention.

separator
[str, optional] non-blank separator used as the column delimiter in the `coordfile`

hms
[bool, optional (Default: False)] Produce output in `HH:MM:SS.S` format instead of decimal degrees?

precision
[int, optional] Number of floating-point digits in output values

output
[str, optional] Name of output file with results, if desired

verbose
[bool (Default: False)] Print out full list of transformation results

Returns

ra
[float] Right Ascension of pixel. If more than 1 input value, then it will be a numpy array.

dec
[float] Declination of pixel. If more than 1 input value, then it will be a numpy array.

Notes

This task performs a full distortion-correction coordinate transformation based on all WCS keywords and any recognized distortion keywords from the input image header. The transformation recognizes the conventions for describing distortion implemented as part of the SIP and Paper IV conventions used with `AstroDrizzle`. Input images can be updated to use these conventions through the use of the `updatewcs` module the STWCS package.

See Also

stwcs

Examples

These examples illustrate the syntax that can be used to run the task in a couple of common modes.

1. Convert a single X,Y position (100,100) from an calibrated ACS image (`j8bt06nyq_flt.fits`) into an undistorted sky position (RA,Dec):

```
>>> from drizzlepac import pixtosky
>>> r,d = pixtosky.xy2rd("j8bt06nyq_flt.fits[sci,1]", 100, 100)
```

2. Convert a list of X,Y positions from the file `'xyfile.dat'` for a calibrated ACS image (`j8bt06nyq_flt.fits`) into undistorted sky positions and write out the result to the file 'radec.dat':

```
>>> r,d = pixtosky.xy2rd("j8bt06nyq_flt.fits[sci,1]", coordfile="xyfile.dat
↪",
...       output="radec.dat")
```

3. The set of X,Y positions from `'input_flt.fits[sci,1]'` stored as the 3rd and 4th columns from the ASCII file `'xy_sci1.dat'` will be transformed and written out to `'radec_sci1.dat'` using:

```
>>> from drizzlepac import pixtosky
>>> r,d = pixtosky.xy2rd("input_flt.fits[sci,1]", coordfile='xy_sci1.dat',
...                      colnames=['c3','c4'], output="radec_sci1.dat")
```

drizzlepac.pixtosky.**xy2rd**(*input*, *x=None*, *y=None*, *coords=None*, *coordfile=None*, *colnames=None*, *separator=None*, *hms=True*, *precision=6*, *output=None*, *verbose=True*)

Primary interface to perform coordinate transformations from pixel to sky coordinates using STWCS and full distortion models read from the input image header.

drizzlepac.pixtosky.**help**(*file=None*)

Print out syntax help for running `pixtosky`.

Parameters

file

[str (Default = None)] If given, write out help to the filename specified by this parameter Any previously existing file with this name will be deleted before writing out the help.

5.1.3 skytopix: Coordinate transformation from sky coordinates

This task allows a user to perform coordinate transformations with the full WCS and distortion model on source positions from sky coordinates to the WCS defined by an image. This task serves as a replacement for the `IRAF.STSDAS` task `rd2xy`, albeit with the added capability of understanding the full distortion model provided in the headers of images updated for use with astrodrizzle and tweakreg. `skytopix` transforms source positions given as RA, Dec values into pixel positions on the image based on the WCS information and any recognized distortion keywords contained in the input image header.

Authors

Warren Hack

License

License

Parameters

input

[str] full filename with path of input image, an extension name [‘sci’,1] should be provided if input is a multi-extension FITS file.

Optional Parameters

ra

[string, optional] RA position in either decimal degrees or HMS format (with or without `':'`) like `'19:10:50.337406303'` or `'19 10 50.337406303'`

dec

[string, optional] Dec position in either decimal degrees or HMS format (with or without `':'`) like `'-60:2:22.186557409'` or `'-60 2 22.186557409'`

coordfile

[str, optional] full filename with path of file with sky coordinates

colnames
: [str, optional] comma separated list of column names from 'coordfile' file containing sky coordinates, respectively. Will default to first two columns if None are specified. Column names for ASCII files will use 'c1','c2',... convention.

precision
: [int, optional] Number of floating-point digits in output values

output
: [str, optional] Name of output file with results, if desired

verbose
: [bool] Print out full list of transformation results (default: `False`)

Returns

x
: [float] X position of pixel. If more than 1 input value, then it will be a numpy array.

y
: [float] Y position of pixel. If more than 1 input value, then it will be a numpy array.

Notes

This task performs a full distortion-correction coordinate transformation based on all WCS keywords and any recognized distortion keywords from the input image header. The transformation recognizes the conventions for describing distortion implemented as part of the SIP and Paper IV conventions used with `AstroDrizzle`. Input images can be updated to use these conventions through the use of the `updatewcs` module the STWCS package.

See Also

stwcs

Examples

These examples illustrate the syntax that can be used to run the task in a couple of common modes.

1. **Convert a single sky position (0:22:07.0088,-72:03:05.429)**
   from a calibrated ACS image (j94f05bgq_flt.fits) into a pixel position (X,Y) without using the TEAL GUI:

```
>>> from drizzlepac import skytopix
>>> x,y = skytopix.rd2xy("j8bt06nyq_flt.fits[sci,1]",
...                      '0:22:07.0088', '-72:03:05.429')
```

2. **Convert a list of (undistorted) sky positions from the file,**

'radec.dat' for a calibrated ACS image (j8bt06nyq_flt.fits)
into distorted pixel positions, and write out the result to the file 'xypos.dat' without using the TEAL GUI:

```
>>> x,y = skytopix.rd2xy("j8bt06nyq_flt.fits[sci,1]",
...                      coordfile="radec.dat", output="xypos.dat")
```

drizzlepac.skytopix.**rd2xy**(*input*, *ra=None*, *dec=None*, *coordfile=None*, *colnames=None*, *precision=6*, *output=None*, *verbose=True*)

Primary interface to perform coordinate transformations from pixel to sky coordinates using STWCS and full distortion models read from the input image header.

drizzlepac.skytopix.**help**(*file=None*)

Print out syntax help for running `skytopix`.

Parameters

file
[str (Default = None)] If given, write out help to the filename specified by this parameter Any previously existing file with this name will be deleted before writing out the help.

5.2 Misc Tools

5.2.1 ResetBits Update of Input

This module provides the capability to set a specific set of bit values in the input DQ arrays to zero. This allows a user to reset pixels previously erroneously flagged as cosmic-rays to good for reprocessing with improved alignment or detection parameters. resetbits - A module to set the value of specified array pixels to zero

This module allows a user to reset the pixel values of any integer array, such as the DQ array from an HST image, to zero.

Authors
Warren Hack

License
License

PARAMETERS

filename
[str] full filename with path

bits
[str] sum or list of integers corresponding to all the bits to be reset

Optional Parameters

extver
[int, optional] List of version numbers of the arrays to be corrected (default: None, will reset all matching arrays)

extname
[str, optional] EXTNAME of the arrays in the FITS files to be reset (default: 'dq')

NOTES

This module performs a simple bitwise-and on all the pixels in the specified array and the integer value provided as input using the operation (array & ~bits).

Usage

It can be called not only from within Python, but also from the host-level operating system command line using the syntax:

```
resetbits filename bits [extver [extname]]
```

EXAMPLES

1. The following command will reset the 4096 bits in all the DQ arrays of the file 'input_flt.fits':

   ```
   resetbits input_flt.fits 4096
   ```

   or from the Python command line:

   ```
   >>> import resetbits
   >>> resetbits.reset_dq_bits("input_file_flt.fits", 4096)
   ```

2. To reset the 2,32,64 and 4096 (sum of 4194) bits in the second DQ array, specified as 'dq,2', in the file 'input_flt.fits':

   ```
   resetbits input_flt.fits 4194 2
   ```

   or from the Python command line:

```
>>> import resetbits
>>> resetbits.reset_dq_bits("input_file_flt.fits", 2+32+64+4096, extver=2)
```

drizzlepac.resetbits.**reset_dq_bits**(*input*, *bits*, *extver=None*, *extname='dq'*)

This function resets bits in the integer array(s) of a FITS file.

Parameters

filename
[str] full filename with path

bits
[str] sum or list of integers corresponding to all the bits to be reset

extver
[int, optional] List of version numbers of the DQ arrays to be corrected [Default Value: None, will do all]

extname
[str, optional] EXTNAME of the DQ arrays in the FITS file [Default Value: 'dq']

Notes

The default value of None for the 'extver' parameter specifies that all extensions with EXTNAME matching 'dq' (as specified by the 'extname' parameter) will have their bits reset.

Examples

1. The following command will reset the 4096 bits in all the DQ arrays of the file input_file_flt.fits:

```
reset_dq_bits("input_file_flt.fits", 4096)
```

2. To reset the 2,32,64 and 4096 bits in the second DQ array, specified as 'dq,2', in the file input_file_flt.fits:

```
reset_dq_bits("input_file_flt.fits", "2,32,64,4096", extver=2)
```

drizzlepac.resetbits.**run**(*configobj=None*)

Teal interface for running this code.

5.2.2 pixreplace: Replace pixels which have one value with another value

This task allows a user to replace pixels in an image or set of images which have one value with a new value; for example, replace all NaNs with a value of -999.9. Pixreplace – Replace pixels which have one value with another value

License
License

`drizzlepac.pixreplace.`**`replace`**(*input*, ***pars*)

Replace pixels in `input` that have a value of `pixvalue` with a value given by `newvalue`.

Parameters

input
[str, @-file, list of filenames] Filename(s) of image to be processed.

pixvalue
[float] Pixel value from `input` file to be replaced. [Default: np.nan]

newvalue
[float] New pixel value to use to replace `pixvalue`. [Default: 0.0]

ext
[int, list of ints, None] Extensions from `input` file to process with new pixel values. If None (default), all image extensions (and only image extensions) will be processed.

Notes

It can be called from within Python using the syntax:

```
>>> from drizzlepac import pixreplace
>>> pixreplace.replace('adriz_nanSCI_drz.fits')
or
>>> epar pixreplace
```

Examples

These examples show how this function can be used in a few different ways.

1. Replace all pixels in all extensions which have a value of NaN in 'adriz_nanSCI_drz.fits' with a constant value of 0.0.

   ```
   >>> from drizzlepac import pixreplace
   >>> pixreplace.replace('adriz_nanSCI_drz.fits')
   ```

2. Replace pixels in first extension only which have a value of NaN in both 'j8c061vnq_drc.fits' and 'j8c061nyq_drc.fits' with a constant value of 0.0.

```
>>> pixreplace.replace('j8c061vnq_drc.fits,j8c061nyq_drc.fits', ext=1)
```

3. Replace pixels in first and fourth extensions which have a value of 0.0 in 'adriz_nanSCI_drz.fits' with a value of -999.

```
>>> pixreplace.replace('adriz_nanSCI_drz.fits',pixvalue=0.0, newvalue=-
↪999, ext=[1,4])
```

drizzlepac.pixreplace.**help**(*file=None*)

Print out syntax help for running `pixreplace`.

Parameters

file

[str (Default = None)] If given, write out help to the filename specified by this parameter Any previously existing file with this name will be deleted before writing out the help.

5.2.3 Photometric equalization for AstroDrizzle

A tool to adjust data values of images by equalizing each chip's PHOTFLAM value to a single common value so that all chips can be treated equally by `AstroDrizzle`.

Authors

Mihai Cara

License

License

drizzlepac.photeq.**photeq**(*files='*_flt.fits'*, *sciext='SCI'*, *errext='ERR'*, *ref_phot=None*, *ref_phot_ext=None*, *phot_kwd='PHOTFLAM'*, *aux_phot_kwd='PHOTFNU'*, *search_primary=True*, *readonly=True*, *clobber=False*, *logfile='photeq.log'*)

Adjust data values of images by equalizing each chip's PHOTFLAM value to a single common value so that all chips can be treated equally by `AstroDrizzle`.

Parameters

files

[str (Default = `'*_flt.fits'`)] A string containing one of the following:

- a comma-separated list of valid science image file names, e.g.: `'j1234567q_flt.fits, j1234568q_flt.fits'`;
- an @-file name, e.g., `'@files_to_match.txt'`. See notes section for details on the format of the @-files.

Note: Valid science image file names are:

- file names of existing FITS, GEIS, or WAIVER FITS files;

- partial file names containing wildcard characters, e.g., `'*_flt.fits'`;
- Association (ASN) tables (must have `_asn`, or `_asc` suffix), e.g., `'j12345670_asn.fits'`.

---

sciext
[str (Default = 'SCI')] Extension *name* of extensions whose data and/or headers should be corrected.

errext
[str (Default = 'ERR')] Extension *name* of the extensions containing corresponding error arrays. Error arrays are corrected in the same way as science data.

ref_phot
[float, None (Default = None)] A number indicating the new value of PHOTFLAM or PHOTFNU (set by 'phot_kwd') to which the data should be adjusted.

ref_phot_ext
[int, str, tuple, None (Default = None)] Extension from which the `photeq` should get the reference photometric value specified by the `phot_kwd` parameter. This parameter is ignored if `ref_phot` **is not** `None`. When `ref_phot_ext` is `None`, then the reference inverse sensitivity value will be picked from the first `sciext` of the first input image containing `phot_kwd`.

phot_kwd
[str (Default = 'PHOTFLAM')] Specifies the primary keyword which contains inverse sensitivity (e.g., PHOTFLAM). It is used to compute conversion factors by which data should be rescaled.

aux_phot_kwd
[str, None, list of str (Default = 'PHOTFNU')] Same as `phot_kwd` but describes *other* photometric keyword(s) that should be corrected by inverse of the scale factor used to correct data. These keywords are *not* used to compute conversion factors. Multiple keywords can be specified as a Python list of strings: `['PHOTFNU', 'PHOTOHMY']`.

Note: If specifying multiple secondary photometric keywords in the TEAL interface, use a comma-separated list of keywords.

---

search_primary
[bool (Default = True)] Specifies whether to first search the primary header for the presence of `phot_kwd` keyword and compute conversion factor based on that value. This is (partially) ignored when `ref_phot` is not `None` in the sense that the value specified by `ref_phot` will be used as the reference *but* in all images primary will be searched for `phot_kwd` and `aux_phot_kwd` and those values will be corrected (if `search_primary=True`).

readonly
[bool (Default = True)] If `True`, `photeq` will not modify input files (nevertheless, it will convert input GEIS or WAVERED FITS files to MEF and could overwrite

existing MEF files if `clobber` is set to `True`). The (console or log file) output however will be identical to the case when `readonly=False` and it can be examined before applying these changes to input files.

clobber
[bool (Default = False)] Overwrite existing MEF files when converting input WAVERED FITS or GEIS to MEF.

logfile
[str, None (Default = 'photeq.log')] File name of the log file.

Notes

By default, `photeq` will search for the first inverse sensitivity value (given by the header keyword specified by the `phot_kwd` parameter, e.g., PHOTFLAM or PHOTFNU) found in the input images and it will equalize all other images to this reference value.

It is possible to tell `photeq` to look for the reference inverse sensitivity value only in a specific extension of input images, e.g.: 3, ('sci',3), etc. This can be done by setting `ref_phot_ext` to a specific extension. This may be useful, for example, for WFPC2 images: WF3 chip was one of the better calibrated chips, and so, if one prefers to have inverse sensitivities equalized to the inverse sensitivity of the WF3 chip, one can set `ref_phot_ext=3`.

Alternatively, one can provide their own reference inverse sensitivity value to which all other images should be "equalized" through the parameter `ref_phot`.

Note: Default parameter values (except for `files`, `readonly`, and `clobber`) should be acceptable for most HST images.

Warning: If images are intended to be used with `AstroDrizzle`, it is recommended that sky background measurement be performed on "equalized" images as the `photeq` is not aware of sky user keyword in the image headers and thus it cannot correct sky values already recorded in the headers.

Examples

1. In most cases the default parameters should suffice:

```
>>> from drizzlepac import photeq
>>> photeq.photeq(files='*_flt.fits', readonly=False)
```

2. If the re-calibration needs to be done on PHOTFNU rather than PHOTFLAM, then:

```
>>> photeq.photeq(files='*_flt.fits', ref_phot='PHOTFNU',
... aux_phot_kwd='PHOTFLAM')
```

3. If for WFPC2 data one desires that PHOTFLAM from WF3 be used as the reference in WFPC2 images, then:

```
>>> photeq.photeq(files='*_flt.fits', ref_phot_ext=3) # or ('sci',3)
```

drizzlepac.photeq.**help**(*file=None*)

Print out syntax help for running photeq.

Parameters

file

[str (Default = None)] If given, write out help to the filename specified by this parameter Any previously existing file with this name will be deleted before writing out the help.

5.2.4 Updatenpol

updatenpol:

```
Update the header of ACS file(s) with the names of new
``NPOLFILE`` and ``D2IMFILE`` reference files for use with the
AstroDrizzle.
```

Authors

Warren Hack

License

License

drizzlepac.updatenpol.**find_d2ifile**(*flist*, *detector*)

Search a list of files for one that matches the detector specified.

drizzlepac.updatenpol.**find_npolfile**(*flist*, *detector*, *filters*)

Search a list of files for one that matches the configuration of detector and filters used.

drizzlepac.updatenpol.**main**()

This task can be run from the operating system command line with:

```
updatenpol [options] input [refdir]
```

Parameters

input

[str] The specification of the files to be updated, either as a single filename, an ASN table name, or wild-card specification of a list of files.

refdir

[str] The name of the directory containing all the new reference files (*_npl.fits and *_d2i.fits files). If no directory is given, it will look in jref$ by default.

``-h``
Print the help (this text).

``-l``
If specified, copy NPOLFILEs and D2IMFILEs to local directory for use with the input files.

``-i``
If specified, the program will interactively request the exact names of the `NPOLFILE` and `D2IMFILE` reference files to be used for updating the header of each file. The value of 'refdir' will be ignored in interactive mode.

.. warning::
It will ask for the names of the `NPOLFILE` and `D2IMFILE` for EACH separate INPUT file when the option `-i` has been specified.

Notes

The new version of `MultiDrizzle` (`AstroDrizzle`) and `updatewcs` only work with the new `NPOLFILE` reference file for the `DGEO` correction (to replace the use of DGEOFILE). In fact, `AstroDrizzle` has been extensively modified to prompt the user with a very lengthy explanation on whether it should stop and allow the user to update the header or continue without applying the `DGEO` correction under circumstances when the `NPOLFILE` keyword can not be found for ACS.

Examples

1. This command will update all the FLT files in the current directory with the new `NPOLFILE` and `D2IMFILE` reference files found in the 'myjref' directory as defined in the environment:

```
updatenpol *flt.fits myjref$
```

drizzlepac.updatenpol.**run**(*configobj=None*, *editpars=False*)

Teal interface for running this code.

drizzlepac.updatenpol.**update**(*input*, *refdir='jref$'*, *local=None*, *interactive=False*, *wcsupdate=True*)

Updates headers of files given as input to point to the new reference files NPOLFILE and D2IMFILE required with the new C version of MultiDrizzle.

Parameters

input
[string or list]

Name of input file or files acceptable forms:

- single filename with or without directory
- @-file
- association table

- python list of filenames
- wildcard specification of filenames

refdir
[string] Path to directory containing new reference files, either environment variable or full path.

local
[boolean] Specifies whether or not to copy new reference files to local directory for use with the input files.

interactive
[boolean] Specifies whether or not to interactively ask the user for the exact names of the new reference files instead of automatically searching a directory for them.

wcsupdate
[boolean] Specifies whether or not to update the WCS information in this file to use the new reference files.

Notes

Warning: This program requires access to the `jref$` directory in order to evaluate the DGEOFILE specified in the input image header. This evaluation allows the program to get the information it needs to identify the correct NPOLFILE.

The use of this program now requires that a directory be set up with all the new NPOLFILE and D2IMFILE reference files for ACS (a single directory for all files for all ACS detectors will be fine, much like jref). Currently, all the files generated by the ACS team has initially been made available at:

```
/grp/hst/acs/lucas/new-npl/
```

The one known limitation to how this program works comes from confusion if more than 1 file could possibly be used as the new reference file. This would only happen when NPOLFILE reference files have been checked into CDBS multiple times, and there are several versions that apply to the same detector/filter combination. However, that can be sorted out later if we get into that situation at all.

Examples

1. A set of associated images specified by an ASN file can be updated to use the NPOLFILEs and D2IMFILE found in the local directory defined using the `myjref$` environment variable in python using:

```
>>> import updatenpol
>>> updatenpol.update('j8bt06010_asn.fits', 'myref$')
```

2. Another use under Python would be to feed it a specific list of files to be updated using:

```
>>> updatenpol.update(['file1_flt.fits','file2_flt.fits'],'myjref$')
```

3. Files in another directory can also be processed using:

```
>>> updatenpol.update('data$*flt.fits','../new/ref/')
```

drizzlepac.updatenpol.**help**(*file=None*)

Print out syntax help for running updatenpol.

Parameters

file

[str (Default = None)] If given, write out help to the filename specified by this parameter Any previously existing file with this name will be deleted before writing out the help.

PYTHON MODULE INDEX

d

s

INDEX

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

W

X