desispec Documentation

Release 0.51.13

DESI

Nov 17, 2022

Contents

1 Contents 1

2 Indices and tables 167

Python Module Index 169

Index 171

CHAPTER 1

Contents

1.1 Overview

The DESI spectroscopic pipeline is a collection of software designed to efficiently take DESI raw data and produce redshift estimates. This software consists of low-level functions that perform individual processing tasks as well as higher-level tools to collectively run whole steps of the pipeline.

1.1.1 Hardware Constraints

To the extent possible, the DESI instrument has been designed to make analysis straightforward. The spectrographs are isolated from the telescope to improve stability, fiber traces are arranged in bundles which should be independent of each other on the CCD, etc. Because of this, we are able to break up the most expensive analysis steps into small independent pieces, and run those small pieces in parallel.

1.1.2 Computing Infrastructure

The workflow of the spectroscopic pipeline reduces to a set of many tasks that depend on each other. Tasks of different types often have different computational needs and run using a varying number of processes. Although this type of workflow could be run on commercial cloud computing platforms (at significant cost), the primary systems available to the DESI project for analysis are the HPC systems at NERSC. These machines are traditional supercomputers with a high speed interconnect between the nodes. The nodes are lightweight and have no local disk. Instead, all nodes share a common Lustre filesystem. SLURM is used for job scheduling, and the machine is used by a variety of workflows and apps across many projects and science domains.

1.1.3 Software Constraints

Given our instrument hardware and computing infrastructure, the spectroscopic pipeline must:

1. Be able to process DESI data at high concurrency on machines at NERSC. HPC software has further requirements / guidelines:

- Large jobs must not log excessively to stdout/stderr.
- Many processes should not read/write to the same files.
- Python startup time (due to filesystem contention loading shared libraries and modules) must be overcome by some method.

2. Be robust against failures of individual small tasks. A failure of one task should only cause failures of future tasks that depend on those specific outputs. This fault tolerance is handled automatically with something like Spark. On an HPC system we need to track such failures.
3. If an individual step/task fails, it must be possible to determine how that step was run and retry it for debugging.
4. It should be possible to determine the current status of a pipeline and which things have failed.
5. (Eventually) the pipeline should interface with a database for tracking the state of individual tasks, rather than querying the filesystem.

1.2 Installation

The DESI spectroscopic pipeline requires and interfaces with other external software packages. This document assumes that you are setting up a "development" software stack from scratch based on the latest versions of the DESI tools.

1.2.1 External Dependencies

In order to set up a working DESI pipeline, there are several external software packages that must be installed on your system. There are several ways of obtaining these dependencies: using your OS package manager, using a third-party package manager (e.g. macports, etc), or using one of the conda-based Python distributions (e.g. Anaconda from Continuum Analytics).

The list of general software outside the DESI project which is needed for the spectroscopic pipeline is:

- BLAS / LAPACK (OpenBLAS, MKL, Accelerate framework, etc)
- CFITSIO
- BOOST
- requests
- matplotlib
- scipy
- astropy
- fitsio
- pyyaml
- speclite

If you wish to run the pipeline on a cluster, also ensure that you have installed mpi4py (which obviously requires a working MPI installation).

Installing Dependencies on a Linux Workstation

For development, debugging, and small tests it is convenient to install the pipeline tools on a laptop or workstation. If you are using a Linux machine, you can get all the dependencies (except fitsio and speclite, which are pip-installable) from your distribution's package manager. Alternatively, you can install just the non-python dependencies with your package manager and then use Anaconda for your python stack. On OS X, I recommend using macports to get at least the non-python dependencies, and perhaps all of the python tools as well.

Example: Ubuntu GNU/Linux

On an Ubuntu machine, you could install all the dependencies with:

```
%> sudo apt-get install libboost-all-dev libcfitsio-dev \
    libopenblas-dev liblapack-dev python3-matplotlib \
    python3-scipy python3-astropy python3-requests \
    python3-yaml python3-mpi4py

%> pip install --no-binary :all: fitsio speclite iniparser
```

Installing Dependencies on an OS X System

Installing scientific software on OS X is often more difficult than Linux, since Apple is primarily concerned with development of apps using Xcode. The approach described here for installing desispec dependencies seems to get the job done with the fewest steps.

First, install homebrew:

```
%> /usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/
↪master/install)"
```

Now use homebrew to install CFITSIO, BOOST, and OpenMPI:

```
%> brew install cfitsio
%> brew install boost
%> brew install openmpi
```

We are going to use homebrew to install our python stack. Some people prefer Anaconda, but that distribution has several problems on OS X. When installing python, we add the flag to indicate we want the python3 versions as well:

```
%> brew install homebrew/python/mpi4py --with-python3
%> brew install homebrew/python/scipy --with-python3
%> brew install homebrew/python/matplotlib --with-python3
```

The rest of the dependencies we can install with pip:

```
%> pip3 install requests pyyaml iniparser speclite astropy
%> pip3 install --no-binary :all: fitsio
```

Dependencies at NERSC

At NERSC there is already a conda-based python stack and a version of the non-python dependencies installed. You can add the necessary module search path by doing (you can safely add this to your ~/.bashrc.ext):

```
%> module use /global/common/${NERSC_HOST}/contrib/desi/modulefiles
```

and then whenever you want to load the software:

```
%> module load desi-conda
```

1.2.2 DESI Affiliated Dependencies

Now we should have our base software stack set up and next we will install dependencies associated with DESI. For this example, we will install DESI software to our home directory. On NERSC systems, the $HOME directory is not as performant as $SCRATCH in terms of python startup time. However, installing software to $SCRATCH (and the necessary steps to prevent it from being purged) is beyond the scope of this document.

Create a Shell Function

Create a bash function that we will use to load our installed desi software into our environment:

```
desidev () {
    # This is the install location of our desi software.
    # If you are not at NERSC, then change this to something
    # without "NERSC_HOST" in the name.
    desisoft="${HOME}/desi-${NERSC_HOST}"

    # Set environment variables
    export CPATH=${desisoft}/include:${CPATH}
    export LIBRARY_PATH=${desisoft}/lib:${LIBRARY_PATH}
    export LD_LIBRARY_PATH=${desisoft}/lib:${LD_LIBRARY_PATH}
    export PYTHONPATH=${desisoft}/lib/python3.5/site-packages:${PYTHONPATH}

    # Special setup for redmonster
    red="${HOME}/git-${NERSC_HOST}/redmonster"
    export PYTHONPATH=${red}/python:${PYTHONPATH}
    export REDMONSTER_TEMPLATES_DIR=${red}/templates

    # Choose what data files to use- these locations
    # are for NERSC.
    export DESI_ROOT=/project/projectdirs/desi
    export DESIMODEL=${DESI_ROOT}/software/edison/desimodel/master
    export DESI_BASIS_TEMPLATES=${DESI_ROOT}/spectro/templates/basis_templates/v2.2
    export STD_TEMPLATES=${DESI_ROOT}/spectro/templates/star_templates/v1.1/star_
↪templates_v1.1.fits
}
```

Now log out and back in. We should pre-create the python package directory the first time we install things:

```
%> mkdir -p ${HOME}/desi-${NERSC_HOST}/lib/python3.5/site-packages
```

At NERSC, first load our dependencies:

```
%> module load desi-conda
```

and then execute our shell function:

```
%> desidev
```

Install Release Tarballs

Now we are ready to install software to this location. Some packages (HARP), do not currently change rapidly and we can just install them from a released tarball. If you are building on a workstation or laptop, download the latest release of HARP from https://github.com/tskisner/HARP/releases and install:

```
%> cd harp-1.0.1
%> ./configure --disable-python --disable-mpi \
   --prefix="${HOME}/desi-${NERSC_HOST}"
```

Note: At NERSC, the HARP package is already included in the software loaded by the desi-conda module. You do not need to build it at NERSC.

Organize Your Git Clones

For the purposes of this document, we assume that all DESI git clones reside in $HOME/git-$NERSC_HOST. You will need to get the following repos. Some of these are not strictly necessary for the spectroscopic pipeline, but are useful for simulating data as part of the integration tests:

```
%> cd $HOME/git-$NERSC_HOST
%> git clone git@github.com:desihub/desiutil.git
%> git clone git@github.com:desihub/desimodel.git
%> git clone git@github.com:desihub/desitarget.git
%> git clone git@github.com:desihub/desisim.git
%> git clone git@github.com:desihub/specter.git
%> git clone git@github.com:desihub/specex.git
%> git clone git@github.com:desihub/desispec.git
%> git clone git@github.com:desihub/redmonster.git
```

Now we are ready to install the various DESI packages from their git source trees. Let's go into our git directory and create a small helper script which will update your install any time you update your source trees:

```
%> cd $HOME/git-$NERSC_HOST
%> cat install.sh

#!/bin/bash

# This should be your actual install location...
pref="${HOME}/desi-${NERSC_HOST}"

cd specex
make clean
SPECEX_PREFIX=${pref} make -j 4 install
cd ..

for pkg in desiutil desimodel desitarget desisim specter desispec; do
    cd ${pkg}
    python setup.py clean
    python setup.py install --prefix=${pref}
    cd ..
done
```

For the initial install, and any you update your source tree versions, do (make sure the install.sh script is executable):

```
%> ./install.sh
```

Now your DESI software stack is complete. Just run the "desidev" shell function to load everything into your environment, and rerun the install.sh script any time you update your source versions.

1.3 Pipeline Use

The DESI spectroscopic pipeline is used to run real or simulated data through one or more stages of a standard sequence of processing operations. The pipeline is designed to function on a supercomputer (e.g. NERSC) or cluster, but can also run locally for small data tests.

1.3.1 Overview

The starting point of the pipeline is real or simulated raw exposures. These exposures are either arcs, flats, or science exposures. The exposures are grouped by night. Each exposure consists of images from up to 10 spectrographs with 3 cameras (r, b, z) each. The processing "steps" that are defined in the pipeline are:

- **preproc**: (all exposure types) Apply image pre-processing.
- **psf**: (only for arcs) Estimate the PSF.
- **psfnight**: (one per night, only for arcs) Combine all PSF estimates for the night.
- **traceshift**: (only for flats and science) Compute the trace locations in preparation for extractions.
- **extract**: (only for flats and science) Extract the maximum likelihood spectra from the pixel data.
- **fiberflat**: (only for flats) Compute a fiber flat from an extracted continuum lamp exposure.
- **fiberflatnight**: (one per night, only for flats) Build the nightly fiberflat.
- **sky**: (only for science) Apply the fiberflat to sky fibers to compute the sky model.
- **starfit**: (only for science) For each spectrograph, apply fiberflat and sky subtraction to standards and fit the result to stellar models.
- **fluxcalib**: (only for science) Apply the fiberflat and sky subtraction to science fibers and then calibrate against the stellar fits.
- **cframe**: (only for science) Apply the calibration to the extracted frames.
- **spectra**: The calibrated output science spectra are re-grouped into files based on their sky location (healpix pixels).
- **redshift**: The redshifts are estimated from re-grouped spectral files.

For a given pipeline "step", there are frequently many independent processing "tasks" that can be batched together. Each processing task usually has some input dependencies (data files) and generates some outputs. In general, a single task has exactly one output file. This allows reconstruction of the state of the processing from examining the filesystem. The pipeline infrastructure is designed to track the dependencies between tasks as well as the current state of each task. When the pipeline actually does the processing, it generates scripts (either slurm scripts for submission to a queueing system or plain bash scripts) that batch together many tasks.

Example: Imagine you had 5 arc exposures you wanted to estimate the PSF on in one job. Estimating the PSF for one exposure consists of 30 individual tasks (one per spectrograph and camera), so there are 150 tasks in this example. Additionally, each of those tasks can run in parallel using one MPI process per fiber bundle and several threads per process.

For a single set of raw data, we might want to have multiple "data reductions" that use different versions of the pipeline software or use different options for the processing. Each independent reduction of some raw data is called a "production". A "production" on disk consists of a directory hierarchy where the data outputs, logs, and scripts are stored. A database is used to track the dependencies and states of all tasks in a production.

1.3.2 User Interface

As discussed above, a single data processing "production" essentially consists of a database and a directory structure of job scripts, logs, and output data files. The primary user interface for running the pipeline on a specific production is the *desi_pipe* command line tool. This takes a command followed by command-specific options. If you want to write a custom script which controls the pipeline in a particular way, then you can also call the same high-level interface used by *desi_pipe*. This interface is found in the *desispec.pipeline.control* module.

Command Help

An overview of available commands can be displayed with:

```
$> desi_pipe --help

usage: desi_pipe <command> [options]

Where supported commands are (use desi_pipe <command> --help for details):
   (------- High-Level -------)
   create   Create a new production.
   go       Run a full production.
   update   Update an existing production.
   top      Live display of production database.
   status   Overview of production.
   (------- Mid-Level --------)
   chain    Run all ready tasks for multiple pipeline steps.
   cleanup  Reset "running" (or optionally "failed") tasks back to "ready".
   (------- Low-Level --------)
   tasks    Get all possible tasks for a given type and states.
   check    Check the status of tasks.
   dryrun   Return the equivalent command line entrypoint for tasks.
   script   Generate a shell or slurm script.
   run      Generate a script and run it.
   getready Auto-Update of prod DB.
   sync     Synchronize DB state based on the filesystem.
   env      Print current production location.

DESI pipeline control

positional arguments:
  command     Subcommand to run

optional arguments:
  -h, --help  show this help message and exit
```

Creating a Production

The first step to using the pipeline is to create a "production" directory for the data processing outputs:

```
$> desi_pipe create --help

usage: desi_pipe create [options] (use --help for details)

Create a new production

optional arguments:
  -h, --help            show this help message and exit
  --root ROOT           value to use for DESI_ROOT
  --data DATA           value to use for DESI_SPECTRO_DATA
  --redux REDUX         value to use for DESI_SPECTRO_REDUX
  --prod PROD           value to use for SPECPROD
  --force               force DB creation even if prod exists on disk (useful
                        for simulations
  --basis BASIS         value to use for DESI_BASIS_TEMPLATES
  --calib CALIB         value to use for DESI_SPECTRO_CALIB
  --db-sqlite           Use SQLite database backend.
  --db-sqlite-path DB_SQLITE_PATH
                        Override path to SQLite DB
  --db-postgres         Use PostgreSQL database backend. You must correctly
                        configure your ~/.pgpass file!
  --db-postgres-host DB_POSTGRES_HOST
                        Set PostgreSQL hostname
  --db-postgres-port DB_POSTGRES_PORT
                        Set PostgreSQL port number
  --db-postgres-name DB_POSTGRES_NAME
                        Set PostgreSQL DB name
  --db-postgres-user DB_POSTGRES_USER
                        Set PostgreSQL user name
  --db-postgres-authorized DB_POSTGRES_AUTHORIZED
                        Additional PostgreSQL users / roles to authorize
  --nside NSIDE         HEALPix nside value to use for spectral grouping.
```

Before creating a production you should have on hand some information about the data and tools you want to use:

1. The location of the raw data.
2. The location of the "spectro" directory containing various auxiliary files (this is location you want to become $DESI_ROOT for the production).
3. The location of the "top-level" directory where you want to put your productions.
4. The name of your production (which will become a subdirectory).
5. The spectro calibration data from an svn checkout.
6. The basis templates data from an svn checkout.

Here is an example, using some simulated data from a survey validation data challenge:

```
$> desi_pipe create \
--root /project/projectdirs/desi \
--db-postgres \
--data ./desi_test/sim \
--redux ./desi_test/redux \
--prod svdc \
--basis /project/projectdirs/desi/spectro/templates/basis_templates/v3.1 \
--calib /project/projectdirs/desi/spectro/desi_spectro_calib/trunk
```

This creates the production directory and subdirectories for all output data products considering the raw data that exists

at the time you run the command. If you add new raw data to your data directory, see the "update" command below.

Just creating a production does not change anything in your environment and the pipeline has no idea how many productions you have created. In order to "activate" your production and use it for future desi_pipe commands, you must source the setup.sh file. In the example above, you would now do:

```
source ./desi_test/redux/svdc/setup.sh
```

And now all future commands will use this production.

Monitoring a Production

For a quick snapshot of the production you can use the "top" command to display updates on the number of tasks in different states. This is refreshed every 10 seconds. For a single snapshot we can use the "–once" option. Building on our example above:

```
$> desi_pipe top --once
----------------+---------+---------+---------+---------+---------+---------+
   Task Type    | waiting | ready   | running | done    | failed  | submit  |
----------------+---------+---------+---------+---------+---------+---------+
preproc         |       0|      690|        0|        0|        0|        0|
psf             |     180|        0|        0|        0|        0|        0|
psfnight        |      60|        0|        0|        0|        0|        0|
traceshift      |     510|        0|        0|        0|        0|        0|
extract         |     510|        0|        0|        0|        0|        0|
fiberflat       |     180|        0|        0|        0|        0|        0|
fiberflatnight  |      60|        0|        0|        0|        0|        0|
sky             |     330|        0|        0|        0|        0|        0|
starfit         |     110|        0|        0|        0|        0|        0|
fluxcalib       |     330|        0|        0|        0|        0|        0|
cframe          |     330|        0|        0|        0|        0|        0|
spectra         |      53|        0|        0|        0|        0|       NA|
redshift        |      53|        0|        0|        0|        0|       NA|
----------------+---------+---------+---------+---------+---------+---------+
```

Here we see that no tasks have been run yet. The "preproc" tasks are in the "ready" state (their dependencies are met). The remaining tasks are in the "waiting" state, since their dependencies are not yet complete.

Whenever a single task runs, it will write a log specific to that task. This file can always be found in the same place within the production directory (run/logs/night/[night]/). If you re-run a specific task (either because it failed or you simpled wanted to run it again), then the per-task log is overwritten in the same location. The pipeline only tracks the current state of a task from its most recent execution, and the per-task log is the output from that most recent run.

The logs from a "job" (the simultaneous batched execution of many tasks) is stored in a per-job directory located in run/scripts/ and named according to the range of processing steps run in the job, the date and job ID. These logs will contain output about the overall number of tasks that were run, how many tasks succeeded and failed, and any errors due to the scheduling system or runtime environment. A new log directory is created for every job that is submitted.

Processing Data with Minimal Interaction

When doing large-scale processing (or re-processing) of many nights of exposures, it is convenient to have a high-level wrapper that submits many jobs to the queueing system with dependencies between jobs to ensure that the processing happens in the correct sequence. This can be done using the "go" command:

```
$> desi_pipe go --help

usage: desi_pipe go [options] (use --help for details)

Run a full production from start to finish. This will pack steps into 3 jobs
per night and then run redshift fitting after all nights are done. Note that
if you are running multiple nights you should use the regular queue.

optional arguments:
  -h, --help            show this help message and exit
  --nights NIGHTS       comma separated (YYYYMMDD) or regex pattern- only
                        nights matching these patterns will be generated.
  --states STATES       comma separated list of states. This argument is
                        passed to chain (see desi_pipe chain --help for more
                        info).
  --resume              same as --states waiting,ready
  --dryrun              do not submit the jobs.
  --nersc NERSC         write a script for this NERSC system (edison | cori-
                        haswell | cori-knl). Default uses $NERSC_HOST
  --shell               generate normal bash scripts, even if run on a NERSC
                        system
  --nersc_queue NERSC_QUEUE
                        write a script for this NERSC queue (debug | regular)
  --nersc_maxtime NERSC_MAXTIME
                        Then maximum run time (in minutes) for a single job.
                        If the list of tasks cannot be run in this time,
                        multiple job scripts will be written. Default is the
                        maximum time for the specified queue.
  --nersc_maxnodes NERSC_MAXNODES
                        The maximum number of nodes to use. Default is the
                        maximum nodes for the specified queue.
  --nersc_shifter NERSC_SHIFTER
                        The shifter image to use for NERSC jobs
  --mpi_procs MPI_PROCS
                        The number of MPI processes to use for non-NERSC shell
                        scripts (default 1)
  --mpi_run MPI_RUN     The command to launch MPI programs for non-NERSC shell
                        scripts (default do not use MPI)
  --procs_per_node PROCS_PER_NODE
                        The number of processes to use per node. If not
                        specified it uses a default value for each machine.
  --outdir OUTDIR       Put task scripts and logs in this directory relative
                        to the production 'scripts' directory. Default puts
                        task directory in the main scripts directory.
  --debug               debugging messages in job logs
```

There are many options to this command that control things like the NERSC sytem to use, the job submission queue, the maximum runtime and number of nodes to use, etc. By default, jobs are submitted to the regular queue with maximum job sizes and run times given by the limits for that queue. Before using non-default values for these at NERSC, you should read and familiarize yourself with the different queues and their limits in the NERSC online documentation.

If the "–nersc" option is not specified, then bash scripts will be generated. You can use other options to enable the use of MPI in these bash scripts and specify the node sizes and properties.

Continuing our example, we could submit several jobs to process all tasks on the cori-knl nodes with:

```
$> desi_pipe go --nersc cori-knl
```

This will submit 3 jobs per night and a final job to do the spectral regrouping and redshift fitting. If some of these jobs fail for some reason, you can cleanup the production (see the cleanup command below with the "–submitted" option) and then re-run the "go" command with the "–resume" option:

```
$> desi_pipe go --nersc cori-knl --resume
```

Updating a Production

When new raw data arrives in the input data directory, we must add the processing tasks for this new data to our database. This is done using the "update" command:

```
$> desi_pipe update --help

usage: desi_pipe update [options] (use --help for details)

Update a production

optional arguments:
  -h, --help       show this help message and exit
  --nights NIGHTS  comma separated (YYYYMMDD) or regex pattern- only nights
                   matching these patterns will be examined.
  --nside NSIDE    HEALPix nside value to use for spectral grouping.
  --expid EXPID    Only update the production for a single exposure ID.
```

By default, the update command looks across all nights in the raw data. This can be time consuming if you have only added a new night of data or a single exposure. Use the options above to restrict the update to only certain nights or exposures.

Cleaning Up When Jobs Fail

There will always be cases where jobs submitted to a queuing system on a large supercomputer will fail. This could be due to a problem with the scheduler, a problem with the filesystem that makes jobs take longer and run out of time, etc. During the running of a job, the state of individual **tasks** are updated as they complete. Even when a job dies or is killed, any completed tasks are marked as done. However, tasks that were in a "running" state when the job ended need to be reset into the "ready" state. This is done using the "cleanup" command:

```
$> desi_pipe cleanup --help

usage: desi_pipe cleanup [options] (use --help for details)

Clean up stale task states in the DB

optional arguments:
  -h, --help            show this help message and exit
  --failed              Also clear failed states
  --submitted           Also clear submitted flag
  --tasktypes TASKTYPES
                        comma separated list of task types to clean (fibermap,
                        rawdata,preproc,psf,psfnight,traceshift,extract,fiberf
                        lat,fiberflatnight,sky,starfit,fluxcalib,cframe,qadata
                        ,spectra,redshift)
  --expid EXPID         Only clean tasks for this exposure ID.
```

You should only run this command if there are no pipeline jobs from the current production running. Additionally, if you are using the "desi_pipe go" command, then tasks already submitted are ignored in future runs. In that case you must use the "–submitted" option to the cleanup command.

Manually Running Processing Steps

Manually running pipeline steps involves first selecting tasks and then running some set of processing steps on these using all the various NERSC queue options.

TO-DO: Document the commands for all this, including:

- tasks
- dryrun
- check
- script
- run
- chain

1.3.3 When Something Goes Wrong

If a job dies, even if due to an external system issue, it is always good to look at the job logs and verify that everything went well up to the point that it failed. The job logs are organized in the run/scripts directory and named after the steps being run, the date and the job ID. For NERSC jobs, you can browse https://my.nersc.gov to get a list of all jobs you have submitted. After verifying that the job ended due to external factors, you can cleanup (see above) and retry.

A pipeline job usually runs many individual tasks. Each task can succeed or fail independently. A pipeline job might complete successfully (from the viewpoint of the queueing system) even if some individual tasks fail. If all tasks fail, the job will exit with a non-zero exit code so that future jobs with a dependency hold are not launched.

If you have a job where one or more tasks failed, you should examine the logs for that task. As discussed before, the per-task logs are in run/logs.

In an extreme case where you believe the production database is invalid or corrupted, you can force the re-creation of the database using only the files that exist on disk. Ensure that all jobs are killed and then do:

```
$> desi_pipe sync
```

This scans the outputs of the production and generates a new DB from scratch.

1.3.4 Example 1: Large (Re)Processing of Many Exposures

Our in-line example in the usage section shows how "desi_pipe go" can be used to submit sets of jobs (3 per night) in a dependency chain and then a final job to do the spectral regrouping and redshift fitting.

1.3.5 Example 2: Process One Exposure

1.3.6 Example 3: Nightly Processing

TO-DO: Document what happens when the "desi_night" command is triggered by the data transfer.

1.3.7 Example 4: Skipping Steps Using External Tools

If you use some of the DESI "quick" simulation tools to produce uncalibrated frame data (or calibrated frame data) directly, then there is a special step that must be taken. In this scenario, the production database has no idea that you have injected data into the processing chain. The only option is to use a recovery step ("desi_pipe sync") which will scan the production output directories and rebuild the database with your injected files included in the dependencies and marked as "done".

1.4 Pipeline Development and Internals

This section needs re-written completely. See issue #531.

1.5 Coaddition of Spectroperfect Reductions

This document covers coadd implementation details and mock-data tests. For details on the coadd dataflow and algorithms used for combining spectra, refer to DESI-doc-1056.

1.5.1 Implementation

Files

All spectra are grouped by brick. There are three types of brick file under `$DESI_SPECTRO_REDUX/$PRODNAME/bricks/{brickid}/`:

- The brick files contains all exposures of every target that has been observed in the brick, by band.
- The band coadd files contain the coadd of all exposures for each target, by band.
- The global coadd files contain the coadd of band coadds for each target.

All files have the same structure with 4 HDUs:

- HDU0: Flux vectors for each spectrum.
- HDU1: Ivar vectors for each spectrum.
- HDU2: The common wavelength grid used for all spectra.
- HDU4: Binary table of metadata.

See the relevant data model descriptions for details (these are not in synch with the mock data challenge files as of 23-Mar-2015).

Wavelength Grids

All brick files have their wavelength grid in HDU2, as summarized in the table below. Note that we do not use a log-lambda grid for the global coadd across bands, since this would not be a good match to the wavelength resolution at the red ends of r and z cameras. See the note for details.

Band	Min(A)	Max(A)	Nbins	Size(A)	Files
b	3579.0	5938.8	3934	0.6	brick, band coadd
r	5635.0	7730.8	3494	0.6	brick, band coadd
z	7445.0	9824.0	3966	0.6	brick, band coadd
all	3579.0	9825.0	6247	1.0	global coadd

Metadata

The brick file and two co-add files all have a table with the same format in HDU4, with one entry per exposure of each object observed in the brick. Most of its columns are copied directly from the exposure fibermaps, except for the last three columns which identify the exposure and offset of the object's spectrum in the other HDUs. The table columns are listed below.

Name	Description
FIBER	Fiber ID [0-4999]
POSITIONER	Positioner ID [0-4999]
SPECTROID	Spectrograph ID [0-9]
TARGETID	Unique target ID
TARGETCAT	Name/version of the target catalog
OBJTYPE	Target type [ELG, LRG, QSO, STD, STAR, SKY]
LAMBDAREF	Reference wavelength at which to align fiber
TARGET_MASK0	Targeting bit mask
RA_TARGET	Target right ascension [degrees]
DEC_TARGET	Target declination [degrees]
X_TARGET	X on focal plane derived from (RA,DEC)_TARGET
Y_TARGET	Y on focal plane derived from (RA,DEC)_TARGET
X_FVCOBS	X location observed by Fiber View Cam [mm]
Y_FVCOBS	Y location observed by Fiber View Cam [mm]
X_FVCERR	X location uncertainty from Fiber View Cam [mm]
Y_FVCERR	Y location uncertainty from Fiber View Cam [mm]
RA_OBS	RA of obs from (X,Y)_FVCOBS and optics [deg]
DEC_OBS	dec of obs from (X,Y)_FVCOBS and optics [deg]
MAG	magitude
FILTER	SDSS_R, DECAM_Z, WISE1, etc.
NIGHT	Date string for the night of observation YYYYMMDD
EXPID	Integer exposure number
INDEX	Index of this object in other HDUs

Programs

The following programs are used to implement the coadd part of the pipeline:

- *desi_make_bricks*: Create brick files from all exposures taken in one night. Reads exposures from cframe files and adds metadata from the exposure fibermap.
- *desi_update_coadds*: Update the coadds for a single brick. Reads exposures from brick files and writes the corresonding band coadd and global coadd files.

An additional program *desi_inspect* displays the information and creates a plot summarizing the coadd results for a single target.

Benchmarks

The rate-limiting step for performing coadds is the final conversion from *Cinv* and *Cinv_f* to *flux*, *ivar* and *resolution* in `desispec.coaddition.Spectrum.finalize()`. The computation time is dominated by one operation: solving the eigenvalue program for a large symmetric real-valued matrix using `scipy.linalg.eigh()`. The computation also involves inverting a real-valued resolution matrix using `scipy.linalg.inv()`, but this is relatively fast.

For a typical r-band coadd on a 2014 MacBook Pro, the total time for a single target is about 12 seconds, dominated by *eigh* (10.6s) and *inv* (1.0s). The time for a global coadd will be longer because of the larger matrices involved.

Interactive tests run on edison@nesrc indicate that it takes about 20s for each single-band coadd and 90s for the global coadd, for a total of about 150s per target. Note that the coadd step can be parallelized across bricks to reduce the wall-clock time required to process an exposure. The biggest speed improvement would likely come from using a sparse matrix eigensolver, or adjusting the algorithm to be able to use an incomplete set of eigenmodes (the `scipy.sparse.linalg.eigsh()` function can not calculate the full spectrum of eigenmodes).

Notes

- The brick filenames have the format `brick-{band}-{expid}.fits`, where `band` is one of [rbz], which differs from the current data model (which is missing the `{band}`).
- Bricks contain a single wavelength grid in HDU2, the same as current CFRAMES, but different from the CFRAME data model (where HDU2 is a per-object mask).
- The NIGHT column in HDU4 has type i4, not string. Is this a problem?
- The 5*S10 FILTER values in the FIBERMAP are combined into a single comma-separated list stored as a single S50 FILTER value in HDU4 of the brick file. This is a workaround until we sort out issues with astropy.io.fits and cfitsio handling of 5*S10 arrays.
- The mock resolution matrices do not have np.sum(R,axis=1) == 1 for all rows and go slightly negative in the tails.
- The wlen values in HDU2 have some roundoff errors, e.g., z-band wlen[-1] = 9824.0000000014425
- Masking via ivar=0 is implemented but not well tested yet.
- We need a way to programmatically determine the brick name given a target ID, in order to locate the relevant files. Otherwise, target ID is not a useful way to define a sample (a la plate-mjd-fiber or ThingID) and an alternative is needed for downstream science users.
- The global coadd sometimes find negative eigenvalues for Cinv or a singular R.T. These cases need to be investigated.

1.5.2 Mock Data Tests

Warning: The description of environment setup and installation below may be out of date.

DESI Environment

Ssh to edison.nersc.gov (remember to use *ssh -A* to propagate your keys for github access) and:

```
source /project/projectdirs/desi/software/modules/desi_environment.sh
```

Installation

Clone the git package and select the co-add development branch (which should soon be merged into the master branch, making the last command unecessary):

```
git clone git@github.com:desihub/desispec.git
cd desispec
git checkout \#6
```

Per-Login Setup

Manually set paths for using this installation (assuming *bash*):

```
cd desispec
export PATH=$PWD/bin:$PATH
export PYTHONPATH=$PWD/py:$PYTHONPATH
```

Set pipeline paths:

```
export DESI_SPECTRO_REDUX=$DESI_ROOT/spectro/redux
export PRODNAME=sjb/cedar2a
export DESI_SPECTRO_SIM=$DESI_ROOT/spectro/sim
export DESI_SPECTRO_DATA=$DESI_SPECTRO_SIM/alpha-5
```

Run Tests

Convert mocks cframes and fibermaps into brick files using:

```
rm -rf $DESI_SPECTRO_REDUX/$PRODNAME/bricks
desi_make_bricks.py --night 20150211 --verbose
```

Note that the code is not yet smart enough to do the right thing for exposures that have already been added to brick files, hence the *rm* command above.

Update coadds for a single brick:

```
rm -rf $DESI_SPECTRO_REDUX/$PRODNAME/bricks/3587m010/coadd*
desi_update_coadds.py --brick 3587m010 --verbose
```

Look at a single target in this brick (this is an LRG):

```
desi_inspect.py --brick 3587m010 --target 3640213155238558158 --verbose
```

Inspect a brick file in iPython using, e.g.:

```
import os,os.path
import astropy.io.fits as fits
from astropy.table import Table
brick = fits.open(os.path.join(os.getenv('DESI_SPECTRO_REDUX'),os.getenv('PRODNAME'),
↪'bricks','3587m010','brick-r-3587m010.fits'))
info = Table.read(brick,hdu=4)
print info
plt.errorbar(x=brick[2].data,y=brick[0].data[0],yerr=brick[1].data[0]**-0.5)
```

Run unit tests:

```
python -m desispec.resolution
```

1.6 Quality Assurance

1.6.1 Overview

The DESI spectroscopic pipeline includes a series of routines that monitor the quality of the pipeline products and may be used to inspect outputs across exposures, nights, or a full production.

1.6.2 Expose QA

Here is the magic to expose a set of QA products made at NERSC to the world:

1. cp -rp QA into www area :: /project/projectdirs/desi/www
2. fix_permissions.sh -a QA [This may no longer be necessary]

These are then exposed at https://portal.nersc.gov/cfs/desi/rest_of_path

1.6.3 Scripts

desi_qa_frame

Generate the QA for an input frame file. The code can be written anywhere and the output is written to its "proper" location.

usage

Here is the usage:

```
usage: desi_qa_frame [-h] --frame_file FRAME_FILE [--reduxdir PATH]
                     [--make_plots]

Generate Frame Level QA [v0.4.2]

optional arguments:
  -h, --help            show this help message and exit
  --frame_file FRAME_FILE
                        Frame filename. Full path is not required nor desired.
  --reduxdir PATH       Override default path ($DESI_SPECTRO_REDUX/$SPECPROD)
                        to processed data.
  --make_plots          Generate QA figs too?
```

examples

Generate the QA YAML file:

```
desi_qa_frame --frame_file=frame-r7-00000077.fits
```

Generate the QA YAML file and figures:

```
desi_qa_frame --frame_file=frame-r7-00000077.fits --make_plots
```

desi_qa_exposure

Generates Exposure level QA. The current implementation is only for the flat flux.

usage

Here is the usage:

```
usage: desi_qa_exposure [-h] --expid EXPID [--qatype QATYPE]
                        [--channels CHANNELS] [--reduxdir PATH] [--rebuild]
                        [--qamulti_root QAMULTI_ROOT] [--slurp SLURP]

Generate Exposure Level QA [v0.5.0]

optional arguments:
  -h, --help            show this help message and exit
  --expid EXPID         Exposure ID
  --qatype QATYPE       Type of QA to generate [fiberflat, s2n]
  --channels CHANNELS   List of channels to include. Default = b,r,z]
  --reduxdir PATH       Override default path ($DESI_SPECTRO_REDUX/$SPECPROD)
                        to processed data.
  --rebuild             Regenerate the QA files for this exposure?
  --qamulti_root QAMULTI_ROOT
                        Root name for a set of slurped QA files (e.g.
                        mini_qa). Uses $SPECPROD/QA for path
  --slurp SLURP         Root name for slurp QA file to add to (e.g. mini_qa).
                        Uses $SPECPROD/QA for path
```

fiberflat

Generate QA on the fiber flat across the exposure for one or more channels.:

```
desi_qa_exposure --expid=96 --qatype=fiberflat
```

desi_qa_skyresid

This script examines sky subtraction resdiuals for an exposure, night or production.

usage

Here is the usage:

```
usage: desi_qa_skyresid [-h] [--reduxdir PATH] [--expid EXPID] [--night NIGHT]
                        [--channels CHANNELS] [--prod] [--gauss]
                        [--nights NIGHTS]

Generate QA on Sky Subtraction residuals [v0.4.2]
```

(continues on next page)

(continued from previous page)

```
optional arguments:
  -h, --help            show this help message and exit
  --reduxdir PATH       Override default path ($DESI_SPECTRO_REDUX/$SPECPROD)
                        to processed data.
  --expid EXPID         Generate exposure plot on given exposure
  --night NIGHT         Generate night plot on given night
  --channels CHANNELS   List of channels to include
  --prod                Results for full production run
  --gauss               Expore Gaussianity for full production run
  --nights NIGHTS       List of nights to limit prod plots
```

Exposure

Generate a plot of the sky subtraction residuals for an input Exposure ID. e.g.

```
desi_qa_sky --expid=123
```

Production

Generate a plot of the sky subtraction residuals for the Production. If reduxdir is not provided, then the script will use the $SPECPROD and $DESI_SPECTRO_REDUX environemental variables. Simply called:

```
desi_qa_sky --prod
```

Gaussianity

Examine whether the residuals are distributed as Gaussian statistics. Here is an example:

```
desi_qa_sky --gauss
```

desi_qa_night

This script is used to analyze the QA outputs from a given night. Note that we use desi_qa_prod (below) to generate the QA YAML files.

usage

Here is the usage:

```
usage: desi_qa_night [-h] [--expid_series] [--bright_dark BRIGHT_DARK]
                     [--qaprod_dir QAPROD_DIR] [--specprod_dir SPECPROD_DIR]
                     [--night NIGHT]

Generate/Analyze Production Level QA [v0.5.0]

optional arguments:
  -h, --help            show this help message and exit
```

(continues on next page)

(continued from previous page)

```
  --expid_series        Generate exposure series plots.
  --bright_dark BRIGHT_DARK
                        Restrict to bright/dark (flag: 0=all; 1=bright;
                        2=dark; only used in time_series)
  --qaprod_dir QAPROD_DIR
                        Path to where QA is generated. Default is qaprod_dir
  --specprod_dir SPECPROD_DIR
                        Path to spectro production folder. Default is
                        specprod_dir
  --night NIGHT         Night; required
```

Current recommendation

First generate the QA for the given night with desi_qa_prod, e.g.:

```
desi_qa_prod --make_frameqa 1 --specprod_dir /global/projecta/projectdirs/desi/
↪spectro/redux/daily --night 20200224 --qaprod_dir /global/projecta/projectdirs/desi/
↪spectro/redux/xavier/daily/QA --slurp
```

Then generate the Night plots:

```
desi_qa_night --specprod_dir /global/projecta/projectdirs/desi/spectro/redux/daily --
↪qaprod_dir /global/projecta/projectdirs/desi/spectro/redux/xavier/daily/QA --night␣
↪20200224 --expid_series
```

desi_qa_prod

This script is used to both generate and analyze the QA outputs for a complete production.

usage

Here is the usage:

```
usage: desi_qa_prod [-h] [--make_frameqa MAKE_FRAMEQA] [--slurp] [--remove]
                    [--clobber] [--channel_hist CHANNEL_HIST]
                    [--time_series TIME_SERIES] [--bright_dark BRIGHT_DARK]
                    [--html] [--qaprod_dir QAPROD_DIR] [--S2N_plot]
                    [--ZP_plot] [--xaxis XAXIS]

Generate/Analyze Production Level QA [v0.5.0]

optional arguments:
  -h, --help            show this help message and exit
  --make_frameqa MAKE_FRAMEQA
                        Bitwise flag to control remaking the QA files (1) and
                        figures (2) for each frame in the production
  --slurp               slurp production QA files into one?
  --remove              remove frame QA files?
  --clobber             clobber existing QA files?
  --channel_hist CHANNEL_HIST
                        Generate channel histogram(s)
  --time_series TIME_SERIES
```

(continues on next page)

(continued from previous page)

```
                        Generate time series plot. Input is QATYPE-METRIC,
                        e.g. SKYSUB-MED_RESID
  --bright_dark BRIGHT_DARK
                        Restrict to bright/dark (flag: 0=all; 1=bright;
                        2=dark; only used in time_series)
  --html                Generate HTML files?
  --qaprod_dir QAPROD_DIR
                        Path to where QA is generated. Default is qaprod_dir
  --S2N_plot            Generate a S/N plot for the production (vs. xaxis)
  --ZP_plot             Generate a ZP plot for the production (vs. xaxis)
  --xaxis XAXIS         Specify x-axis for S/N and ZP plots
```

frameqa

One generates the frame QA, the YAML and/or figure files with the –make_frameqa flag. These files are created in a folder tree QA/ that is parallel to the exposures and calib2d folders.:

```
desi_qa_prod --make_frameqa=1  # Generate all the QA YAML files
desi_qa_prod --make_frameqa=2  # Generate all the QA figure files
desi_qa_prod --make_frameqa=3  # Generate YAML and figures
```

The optional –remove and –clobber flags can be used to remove/clobber the QA files.

slurp

By using the –slurp flag, one generates a full YAML file of all the QA outputs:

```
desi_qa_prod --slurp   # Collate all the QA YAML files into a series of JSON files,␣
↪one per night
desi_qa_prod --slurp --remove  # Collate and remove the individual files
```

html

A set of static HTML files that provide simple links to the QA figures may be generated:

```
desi_qa_prod --html  # Generate HTML files
```

The top-level QA file (in the QA/ folder) includes any PNG files located at the top-level of that folder.

Channel Histograms

Using the –channel_hist flag, the script will generate a series of histogram plots on default metrics: FIBERFLAT: MAX_RMS, SKYSUB: MED_RESID, FLUXCALIB: MAX_ZP_OFF:

```
desi_qa_prod --channel_hist
```

Time Series Plot

Using the –time_series input with a *qatype* and *metric* produces a Time Series plot of that metric for all nights/exposures/frames in the production, by channel, e.g.:

```
desi_qa_prod --time_series=SKYSUB-MED_RESID
desi_qa_prod --time_series=FLUXCALIB-ZP
```

By default, these files are placed in the QA/ folder in the $DESI_SPECTRO_REDUX/$SPECPROD folder.

<S/N> Plot

Generate a plot of <S/N> for a standard set of fiducials – object type at a given magnitude in a given channel (e.g. ELG, 23 mag in channel r). The x-axis is controlled by the *–xaxis* option and may be MJD, texp (exposure time), or expid. Here is a sample call:

```
desi_qa_prod --S2N_plot --xaxis texp
```

ZP Plot

Similar to the <S/N> plot above but for the Zero Point calculated in the three channels. Again, *–xaxis* controls the abscissa axis. An example:

```
desi_qa_prod --ZP_plot --xaxis texp
```

1.7 desispec Change Log

1.7.1 0.52.0 (unreleased)

- No changes yet.

1.7.2 0.51.13 (2022-02-28)

- `desi_zcatalog --patch-missing-ivar-w12` option to patch missing FLUX_IVAR_W1/W2 values that weren't propagated by early fiberassign (PR #1717).

1.7.3 0.51.12 (2022-02-23)

- Remove unnecessary `specter.psf` import, which also allows `desispec` utilities to be imported without explicitly requiring `specter` (PR #1709).
- Let plot_spectra show errors even with –rebin (PR #1714, #1708).
- add SPGRPVAL to desi_zcatalog for custom coadds/redshift group tracking (PR #1712).
- desi_assemble_tilepix replacement for desi_map_tilepix (PR #1713).
- fix read_tile_spectra for group='cumulative' (PR #1696).

1.7.4 0.51.11 (2022-02-21)

- qso_qn afterburner fix for case when all inputs are masked (PR #1704).

1.7.5 0.51.10 (2022-02-18)

- tile-qa avoid divide-by-zero crash on unknown goaltime (PR #1698).
- propagate HEALPIX into zpix redshift catalogs (PR #1699).
- Fix GOALTIME in exposures FRAMES HDU; avoid EFFTIME_ETC NaN (PR #1701).

1.7.6 0.51.9 (2022-02-17)

Fuji cleanup bugfixes.

- tile-qa goaltime special case for tiles 80715,80718 (PR #1689).
- qso afterburner output breadcrumb file if missing input camera (PR #1691).
- fix unwisebrightblue PROGRAM=other not bright (PR #1694).
- fix tsnr afterburner GOALTIME exp vs. tile consistency (PR #1694).
- fix plot_spectra with astropy 5 (PR #1695).

1.7.7 0.51.8 (2022-02-13)

Bugfixes for Fuji; all impacted tiles/nights/healpix rerun with this tag, remaining tiles/nights/healpix are backwards compatible.

- Set specmask BADFIBER only for impacted cameras, not all BRZ (PRs #1674 (master), #1678 (fuji))
- Fix assemble_fibermap with input NaNs for astropy 5.0 (PR #1681).
- Use only 120s flats for nightlyflat (PR #1682).
- Add desi_purge_tilenight script (PR #1683).
- Fix healpix input expid bookkeeping (PR #1684).

1.7.8 0.51.7 (2022-02-10)

- fix tile-qa expid bookkeeping (PR #1670).
- desi_tile_qa exposure/night bookkeeping fix (PR #1672).
- Fix tsnr_afterburner exposure files SURVEY column (PR #1675).

1.7.9 0.51.6 (2022-02-09)

Used for Fuji healpix redshifts and cleanup of failed tile-qa. Backwards compatible with previously run steps.

- Make tile-qa robust to missing cameras (PR #1665)
- Refactor healpix redshifts workflow (PR #1668)

1.7.10 0.51.5 (2022-02-07)

Used for processing nightly biases for Fuji nights 20210331 and 20210422, and Guadalupe night 20210629. Backwards compatible with other nights.

- Additional desi_compute_nightlybias options for flexibility on which ZEROs to use (PR #1662).

1.7.11 0.51.4 (2022-02-04)

Pipelining fix for Fuji; previously run impacted nights will be resubmitted.

- Fix stdstar camword logic when input exposures have different cameras available (PR #1658).

1.7.12 0.51.3 (2022-01-31)

NOTE: this tag fixes a crash, but also produces slightly different humidity correction for a small set Fuji/Guadalupe exposures already run with an earlier tag. This note will be updated if those exposures are reprocessed with this tag.

- Fix fiberflat humidity correction indexing bug when hear (but not at) upper limit of model humidity range (PR #1642).

1.7.13 0.51.2 (2022-01-27)

Fuji bug fixes (impacted nights will be re-run; nights run with earlier tags not impacted)

- fix pipeline bug on nights with multiple 300s darks (PR #1635).
- fix `io.findfile(..., groupname='perexp')` (PR #1637).

1.7.14 0.51.1 (2022-01-26)

Fuji bugfix tag made from the fuji branch mid-processing. These changes fix crashes but do not impact any data that were already successfully run.

- Updated desi_find_badpos script to cross-reference flagged petals against existing bad-exposure tables.
- Fix fiberflat crash when almost all input data are masked for a fiber (PR #1629).
- Fix tile QA for cases when input fiberassign file is not gzipped (PR #1630).
- Fix zcat stacking typo (PR #1633).

1.7.15 0.51.0 (2022-01-24)

This version will be used for Fuji.

Algorithm update:

- Normalize fiberflat variation of each fiber for humidity correction (PR #1621).

Metadata tracking updates:

- Add BADAMP[BRZ] bits to QAFIBERSTATUS (PR #1610).
- specgroup metadata in spectra, coadd, zcat files (PR #1618).

New and fixed scripts / functions:

- Add desispec.zcatalog.find_primary_spectra (PR #1609).
- Add desispec.tile_qa.get_tilecov tile coverage plotting (PRs #1613, #1617).
- Fix bookkeeping of nights and tiles in coadds (issue #1349) and enable coadding of previously coadded cframe files (issue #1359) (PR #1616).
- Ensure `tilepix.fits` only contains healpixels with reduced data (issue #1374). Also fix issues #1373 and #1379 (PR #1614).
- Add desi_find_badpos script to find exp-petals with catastrophic positioning (PR #1620).

1.7.16 0.50.1 (2022-01-20)

- Modification extname in QN afterburner outputs. Add flag in desi_qso_catalog_maker for retro-compatibility (PR #1597).
- Outlier rejection in skycor PCA (PR #1598).
- Include FAILED jobs in resubmissions (PR #1602).
- tile-qa updates for special tiles (PR #1603).
- Better masking behavior for BADCOLUMN (PR #1605).
- Add DEPNAM/DEPVER to fibermap HDUs (PR #1607).
- desi_run_night enhancements and bug fixes (PR #1608).

1.7.17 0.50.0 (2022-01-16)

Used for the f5 test run part II.

- Algorithmic changes:
  - Use sky fibers to correct for serial readout CTE problems (PR #1571).
  - Bug fixes for fiberflat_vs_humidity when near edge of model range (PRs #1589, #1594).
- Miscellaneous:
  - Approximate for missing turbulence corrections in FIBER_RA/DEC during Dec 2021 (PR #1539).
  - nightqa improved selection of which 5min dark was used (PR #1584).
  - findfile(tileqapng) bugfix for tile/night QA (PR #1585).
  - night QA 5min DARK expid selection bug fix (PR #1586).
  - fix qprod non-empty fibermaps (PR #1587).
  - desi_update_specstatus –lastnight and –all options (PR #1588).
  - better desi_proc error tracking (PR #1590).
  - assemble_fibermap fail faster on invalid inputs (PR #1592).

1.7.18 0.49.1 (2022-01-10)

Used for the f5 test run part I.

- Fix propagation of per-camera keywords into per-camera fibermap (commit #0c7aa720)

1.7.19 0.49.0 (2022-01-10)

- Major algorithmic updates:
  - Fit CCD residual background between blocks of fiber traces (PR #1551, #1581).
  - Correction for fiberflat variations with humidity (PR #1565).
- Other algorithmic updates:
  - Updated readnoise estimation when doing overscan per row (PR #1564).
  - Remove average of overscan cols to overscan rows (PR #1575).
  - Avoid false positive bad column mask on noisy input data (PR #1579).
  - Improvements to desi_interpolate_fiber_psf (PR #1557).
- Miscellaneous:
  - run nightlybias on nights without a dark (PR #1553).
  - nightqa petal n(z) support programs with no tiles (PR #1549, #1554).
  - Fix GitHub tests fitsio/numpy incompatibility issues (PR #1566).
  - Bad readnoise PSF failure robustness (PR #1568).
  - Bad exposure bookkeeping for darks and biases (PR #1570).
  - don't flag a tile as archived if archiving failed (PR #1572).
  - Tile QA plotting updates (PR #1577,)
  - Fix fitsverify errors when creating preproc files (PR #1582).
  - Added `assemble_fibermap --tilepix` option (PR #1583)

1.7.20 0.48.1 (2021-12-21)

Used for the f4 test run (albeit pre-tag).

- Fix deprecation warnings for latest numpy (PR #1525).
- Support astropy 5.x masked columns (PR #1526).
- More robust when fiberassign file is in earlier expid (PR #1529, #1536).
- Approximate FP coords if missing from coordinates file (PR #1532).
- desi_run_night –surveys option (PR #1533).
- night_qa v3 (PR #1535).
- Update and standardize exposure tables (PR #1537).
- Fix desi_proc spexec wrapper for single camera (PR #1540).
- Fix desi_proc MPI logic bug if missing PSF input (PR #1542).
- Change nightlybias OSTEP to be local instead of global diff (PR #1543).
- Run nightlybias for all cameras (PR #1546).
- Tune nightlybias running logic (PR #1547).
- Add emlinefit afterburner (PR #1386).

1.7.21 0.48.0 (2021-12-10)

- Fix TSNR afterburner "unknown" entries (PR #1495).
- Tile QA skip n(x) comparison for backup program (PR #1497).
- assemble_fibermap robust to missing guider EXPTIME (PR #1498).
- update job dependencies to be afterok instead of afterany (PR #1502).
- Add desi_night_qa (PR #1503, #1522).
- desi_tile_vi only show main dark/bright tiles by default (PR #1505).
- Add support for short QA flats in pipeline (PR #1507, #1517).
- Revise size of KNL batch jobs for arc fits (PR #1508, #1521).
- fix proctable entries with 0-length arrays (PR #1509).
- flag FIBER_X/Y==0 as FIBERSTATUS MISSING (PR #1514).
- Separate tile QA from tile archiving (PR #1519).
- Improve pipeline operations on KNL (PR #1523).

1.7.22 0.47.1 (2021-11-24)

- New desi_resubmit_queue_failures script (PR #1482).
- fix CAMERA column name in stdstars INPUT_FRAMES HDU (PR #1484).
- raise exception when no valid sky fibers (PR #1486, #1488).
- Update qso_catalog_maker to include all targets (PR #1487).
- Fix TSNR2 calculation and exposure QA for 2-amp readout (PR #1489).
- Exclude masked pixels in stdstar RMS QA (PR #1490).
- Support for very early fiberassign files in `assemble_fibermap()` (PR #1492).
- desi_tile_vi –qastatus option (PR #1493).

1.7.23 0.47.0 (2021-11-11)

- tsnr_afterburner support for old HDU names (PR #1403).
- Tiles tables and QA cleanup (PRs #1406, #1407, #1409, #1410, #1430, #1442, #1445, #1449, #1458, #1475).
- Support averaging PSFs with different wavelength ranges (PR #1411).
- QSO afterburner add blank file when no targets to write (PR #1412).
- Update airmass dependence for exposure quality cuts (PR #1413).
- Only use good sky fibers for sky model (PR #1414).
- Switch EFFTIME_SPEC to be based upon LRG instead of ELG (PR #1417).
- specex job scheduler for improved performance (PR #1418).
- Add desi_update_tiles_specstatus script (PR #1421).
- Parallelize fiberflatnight and cleanup tempfiles (PR #1427).

- Cleanup bad fiber tracking BROKENFIBERS, BADCOLUMNFIBERS, LOWTRANSMISSIONFIBERS (PR #1429).
- Select calibration stars per exposure across petals (PR #1434).
- QSO afterburner run on all targets, not just QSO (PR #1435).
- Set OMP_NUM_THREADS in batch script (#1437).
- Allow fiberassign svn to have different negative TARGETID (PR #1444).
- Arc jobs restricted to <= 10 nodes even on KNL (PR #1450).
- Sky model bug fix to avoid fake z~4.3 QSO (PR #1452).
- desi_edit_exposure_table useability improvements (PR #1453).
- switch io.read_fibermap to use fitsio to avoid astropy masked columns (PR #1454, #1479)
- daily pipeline runner cache exposures after every new exp (PR #1455).
- Hartmann doors analysis script (PR #1457).
- Ensure consistent output from `assemble_fibermap()` (PR #1458).
- New desi_compute_nightly_bias script (PR #1460).
- Incorporate nightly bias and bad column identification into daily pipeline (PR #1463).
- Add OSTEP metric for variation of overscan per row (PR #1464).
- Add nightly bias and bad column flagging to pipeline (PR #1465, #1467)
- Fix check_for_outputs bug and teach findfile about fit-psf (PR #1469).
- Set fibermap.FIBERSTATUS BADREADNOISE and BADAMP[BRZ] (PR #1472).
- Don't use PSFs with bad amps (PR #1473).
- Support 2-amp readout (PR #1476).
- Use only offset traceshifts when amplifier is masked (PR #1477).
- desi_archive_tilenight to archive nights after tile QA approval (PR #1478).
- processing dashboard useability updates (PR #1480).
- use desitarget.skybricks to check stuck sky locations (PR #1481).

1.7.24 0.46.1 (2021-09-03)

- Restore `desi_zcatalog` backwards compatibility for inputs without a separate EXP_FIBERMAP (PR #1392).
- `tsnr_afterburner` robustness to missing/different columns, e.g. from reprocessed daily exposures (PR #1396).
- Save STDSTAR_FIBERMAP in fluxcalib file recording which stars were used (PR #1400).
- stdstars robust to missing individual frames (PR #1402).

1.7.25 0.46.0 (2021-09-01)

- Detect and flag sky fibers affected by a bright source (PR #1367)
- Adjust spectral traces when computing CCD variance in preprocessing (PR #1368, #1378).
- Detect bad CCD columns in nightly dark; use to mask impacted wavelengths of individual spectra (PR #1371).
- Model CCD readnoise with Poisson noise from the zero exposure dark current (PR #1372).
- Speed up coadd_cameras (PR #1377).
- Improve sky subtraction with PCA model of wavelength and LSF sigma adjustments (PR #1381).
- Swap fibers 3402 and 3429 if input fiber:location map is wrong (PR #1382).
- Tile QA minor fixes (PR #1385).
- Allow fiberassign SVN to override raw data fiberassign (PR #1387).
- Add `desi_run_night --tiles` option (PR #1391).

1.7.26 0.45.3 (2021-07-29)

Everest cleanup PRs; in 21.7e software release.

- Fix "unknown" entries in tsnr/exposures file (PR #1321).
- desi_healpix_redshift options for extra memory (PR #1343).
- desi_zcatalog for new fibermap format (PR #1347).
- Include TILERA, TILEDEC, MJD in tiles/exposures output tables (PR #1348).
- desi_group_spectra –healpix option (PR #1350).
- desi_tile_redshifts –group cumulative bug fixes (PR #1353).
- desi_map_tilepix script (PR #1358).
- merge of above PRs into main/master (PR #1360).

1.7.27 0.45.2 (2021-07-20)

Everest bookkeeping update for sv1 spectra regrouping and sv3 redshift cleanup.

- desi_group_spectra options to filter and group by survey and faprogram, including `desispec.io.meta.faflavor2program` to handle special cases for sv1 (PR #1341).
- desi_healpix_redshifts options for memory usage (PR #1341).

1.7.28 0.45.1 (2021-07-15)

For Everest tile spectra+coadds+redshifts; in 21.7d

- Recompute proccamword from exptable for proecessing dashboard (PR #1340).
- Add support for healpix coadd+redshift jobs (PR #1341).

1.7.29 0.45.0 (2021-07-14)

For Everest tile spectra+coadds+redshifts; in 21.7c

Note: this is a major format change to coadd and redrock (formerly zbest) files:

- FIBERMAP split into FIBERMAP (coadded) + EXP_FIBERMAP (per-exposure)
- zbest files renamed redrock

Requires redrock >= 0.15.0

Details / PRs:

- tune tile QA parameters
- Add desi_qso_catalog_maker (PRs #1322, #1339).
- QSO afterburner cross checks and bug fixes (PRs #1334, #1335)
- Fix exposure table loading typo (PR #1337).
- Rename zbest -> redrock (PR #1338).
- Split coadd FIBERMAP into FIBERMAP + EXP_FIBERMAP (PR #1330).
- QSO afterburners run in desi_tile_redshifts by default (PR #1330).

1.7.30 0.44.2 (2021-07-07)

Intended for Everest science exposures through cframes; in 21.7b.

- Fixed data/qa/ installation.

1.7.31 0.44.1 (2021-07-07)

- Add fibermap PLATE_RA,PLATE_DEC if missing from fiberassign file for sv3 and main survey data model consistency (PR #1331).
- Set FIBERSTATUS POORPOSITION bit when positioner is offset by 30-100 microns. Don't use these for stdstar fitting but otherwise process as normal. (PR #1333).

1.7.32 0.44.0 (2021-07-06)

First tag used for Everest arc/flat calibs; in 21.7a.

- Add QSO afterburners for MgII and QuasarNet (PR #1312).
- Spectra I/O for extra catalog (PR #1313).
- Expand Spectra.select and .update functionality (PR #1319).
- Add optional support for gpu_specter for extractions (PR #993).
- Fix extra_catalog support for grouping by healpix (PR #1325).
- Pipeline progress bug fixes and features (PRs #1326, #1329).

1.7.33 0.43.0 (2021-06-21)

This version was used for QA assessment of the first 315 main survey tiles released for unlocking overlapping tiles. That was done pre-tag under the development version "0.42.0.dev5412".

- Improved stitching of sky spectra from different cameras (PR #1273).
- TSNR updates (PR #1274 and branch PRs #1275, #1277, #1279, #1282, #1283, #1285).
- qproc robust to blank SEEING keyword (PR #1289).
- update SV1-SV3 average throughtput (PR #1291).
- fix x traceshift indexing bug (PR #1292).
- desi_tile_redshifts –run_zqso option (PR #1293).
- pre-write speclog when generating dark model scripts (PR #1300).
- Add spectroscopic QA (PR #1302, #1316).
- Improve pipeline metadata handling and implement QA cuts (PR #1304).
- Check for completely masked fibers in qfiberflat (PR #1306).
- Pipeline robustness when reading ETC values from raw data (PR #1309).
- Adjust exposure QA cuts, cleanup outputs (PRs #1316, #1318).
- Simplified tile QA (PR #1317).
- zmtl using tile QA to set ZWARN bits (PR #1310).
- Look for manifest files in nightly processing (PR #1320).

1.7.34 0.42.0 (2021-05-14)

Requires desiutil >= 3.2.1 for new dust extinction calculations.

- Wrap `bin/desi_proc` and `bin/desi_proc_joint_fit` in functions to facilitate pipeline wrappers (PRs #1242 and #1244).
- Use less restrictive gfaproc extension 2 instead of 3 for EFFTIME_GFA (PR #1245).
- Add MPI to stdstar `match_templates` (PR #1248).
- Updates to `desi_average_flux_calibration` (PR #1252).
- `desi_fit_stdstars --std-targetids` option to override stdstars for testing and custom fields (PR #1257, #1259).
- Launch redshifts automatically as part of pipeline (PR #1260).
- Support stuck positioners assigned to sky locations (PR #1266).
- Use desiutil.dust for extinction including Gaia (PR #1269).
- Fix running instance checking in daily pipeline (PR #1270).

1.7.35 0.41.0 (2021-04-16)

Although most of the Denali production was run with tag 0.40.1, the following updates where made for final steps to recover some missing coadds and make the final tsnr and tiles files:

- Exposure and tiles files updates, including merging GFA data. (PR #1226, #1232, #1236, plus commits directly to master on Apr 12).
- Fix coadds with missing TSNR columns due to missing cameras (PR #1239).

Changes that also occured in the meantime but were not used for Denali processing (they impact earlier steps):

- Flag fibers that are mis-positioned by >100 um as bad. (PR #1233).
- Correct bit flagging and support split exposures with `bin/assemble_fibermap` (PR #1235).
- Also write fibercorr to the fluxcalibration when using low S/N simplified calibration (direct fix to master).

1.7.36 0.40.1 (2020-04-01)

Installation and job submission fixes for Denali; no algorithmic changes.

- fix data installation (PR #1221).
- `desi_tile_redshifts --batch-reservation` fix for Denali run (PR #1222).

1.7.37 0.40.0 (2021-03-31)

First tag for 21.3/Denali run

- Add fiber crosstalk correction (PR #1138).
- Handle missing NIGHT in coadded fibermap (PR #1195).
- Add `desi_tiles_completeness` script with TSNR2-based tile completeness calculations for survey ops (PR #1196, #1200, #1204, #1206, #1212).
- TSNR2 camera coadd fix (PR #1197).
- refactor *desi_tile_redshifts* for more flexibility (PR #1198, #1208, #1211).
- processing dashboard: cache night info (PR #1199).
- speed up sky calculation with different sparse matrices (PR #1209).
- Check file outputs before submitting jobs (PR #1217).
- improve noise of master dark model fit (PR #1219).
- Add workflow hooks for KNL (PR #1220).

1.7.38 0.39.3 (2020-03-04)

Cascades update tag for final catalog creation.

Note: datamodel changes to coadd SCORES and FIBERMAP

- Propagate TSNR2 into coadd SCORES; update coadd FIBERMAP columns (PR #1166)
- `bin/desi_tsnr_afterburner` use pre-calculated TSNR2 from frame files unless requested to recalculate (PR #1167).

1.7.39 0.39.2 (2021-03-02)

Cascades update tag to fix coadd and tSNR crashes, and postfacto tag `desi_spectro_calib` version in desispec module file.

- Processing dashboard useability updates (PR #1152).
- Undo heliocentric correction in throughput analysis not used for production processing (PR #1154).
- Fix coadd crash (PR #1163).
- Fix tSNR alpha<0.8 crash (PR #1164).
- Updated desi_spectro_calib version to 0.2.4.

1.7.40 0.39.1 (2021-02-23)

Cascades update tag to add functionality for using a queue reservation and for debugging, without algorithmically impacting what has already been run with the 0.39.0 tag.

- Add `desi_run_night --reservation` option (PR #1145).
- Fix `desi_process_exposure --no-zero-ivar` option (PR #1146).

1.7.41 0.39.0 (2021-02-16)

Initial tag for Cascades run.

Major updates:

- Update exposure table formats and pipeline workflow (PR #1135, #1139).
- Add template S/N (TSNR) depth calculations (PR #1136).

Smaller updates:

- Propagate fiberassign HDU 0 keywords into fibermap header in addition to `FIBERASSIGN` (HDU 1) keywords (PR #1137).
- `desi_proc_joint_fit` exit with error code if all cameras fail (PR #1140).
- Frame units "electron/Angstrom" instead of "count/Angstrom" (PR #1142).

1.7.42 0.38.0 (2021-02-10)

- Change how specex PSF fitting is called; requires specex>=0.7.0 (PR #1082)

1.7.43 0.37.0 (2021-02-10)

Major updates:

- Support Gaia stdstars (PR #1105, #1109, #1114, #1133).
- Fix cosmics masking in coaddition (PR #1113).
- Improved sky modeling (PR #1125).

Smaller (but important) updates:

- Standardize getting NIGHT from raw data headers (PR #1083, #1120).

- Use acquisition guide file if full guide file isn't available (PR #1084).
- Updates to flux calibration averages used by nightwatch (PR #1085).
- New read_tile_spectra and Spectra class slicing (PR #1107).
- Add token to fix coverage tests (PR #1112).
- Flux calibration robustness for low transmission exposures (PR #1116).
- Apply heliocentric correction to fiberflat (PR #1118).
- Robustness and feature updates to dark model generation (PR #1119, #1123)
- More flexible CCD calibration configuration (PR #1121).
- Processing dashboard useability updates (PR #1127).
- NIGHT int vs. str bugfix in QA (PR #1129).
- Support coaddition of fibermaps with different columns (PR #1130).

1.7.44 0.36.1 (2021-01-04)

- Fix PSF traceshifts when a fiber is completely masked (PR #1080).
- Robust to NaN in desi_average_flux_calibration (commit f1de1ac).
- Increase arc and flat runtimes (commit 7cb294c).

1.7.45 0.36.0 (2020-12-23)

This is the primary tag for the Mt. Blanc spectro pipeline run.

- Major updates:
  - Coadd fluxes in multi-exp standard stars before fitting (PR #1059).
  - New model of CCD pixel-level variance (PR #1062).
  - Adjust sky-line variance based on model chi2 (PR #1062).
- Smaller (but important) updates:
  - Fixes assemble_fibermap for older data (PR #1047, bug introduced in PR #1045).
  - Use EBV instead of MW_TRANSMISSION_G/R/Z from fiberassign (PR #1048).
  - Fallback to using FA_TYPE if no stdstars in (SVn_)DESI_TARGET (PR #1050).
  - Use GitHub Actions for testing instead of Travis (PR #1053).
  - Fix stdstar absolute symlinks (PR #1056).
  - Adjust nodes per job (PR #1056 and #1068).
  - Workflow options for bad exposures and new end-of-cals manifests (PR #1057).
  - stdstar robustness if petal is disabled (PR #1060).
  - improved camera argument parsing (PR #1061).
  - Fix unphysical spike at edge of calibration vectors (PR #1065).
  - Add header keywords for input calib provenance (PR #1069).
  - More logging about stdstar selection cuts (PR #1070).

  - Only uses fiberassign .fits and .fits.gz (but not .fits.orig) (PR #1072).
  - Support "unpositioned" exposures; propagate FIBER_RA/DEC if present (PR #1073).
  - Use desi_spectro_calib tag 0.2.1

1.7.46 0.35.0 (2020-12-11)

- Major updates:
  - New opts to model image variance and improve sky subtraction (PR #1008).
  - Refactor desi_proc and daily processing workflow (PRs #1012, #1014, #1030)
  - New bias+dark model ("non-linear dark y1D") in desi_spectro_calib 0.2.0 (PR #1029)
- Smaller (but important) updates:
  - etc/desispec.modules uses desi_spectro_calib 0.2.0
  - Default saturation 2**16-1; updated keywords (PR #1046).
  - Fix preproc header keyword propagation (PR #1045).
  - Add support for gzipped fiberassign files (PR #1042).
  - Fix tests on single-core machines (PR #1035).
  - *desi_paste_preproc* for future use combining short+long arcs (PR #1034).
  - *desi_proc* more robust to *specex* failures (PR #1033).
  - Add parallelism to *desi_preproc* (PRs #1032, #1036, #1038).
  - Fix specex empty path bug (PR #1031).
  - Better qproc warnings for test slit exposures (PR #1028).
  - *desi_focus* focus scan analysis (PR #1027).
  - Fix/add BUNIT header keyword (PR #1023).
  - Adds *desi_compute_broadband_pixel_flatfield* (PR #1022).
  - Update desi_proc timing logging (PR #1003, #1026).
  - desispec.module sets MPICH_GNI_FORK_MODE=FULLCOPY for MPI+multiprocessing (PR #1007).
  - Fix dark CCD calibration corrections (PR #1002).

1.7.47 0.34.7 (2020-09-01)

- Switch desi_proc to use fitsio instead of astropy.io.fits to work around incompatibility between mpi4py and astropy 4 (PR #996).

1.7.48 0.34.6 (2020-08-04)

- Extend runtime limit for spectra regrouping task (hotfix to master).

1.7.49 0.34.5 (2020-08-04)

- Faster desi_zcatalog merging with target table (PR #994).
- Python 3.8 support (PR #990).
- Astropy 4.x support (PR #989).
- Update CCD mask generation code (PR #987).
- Update desispec.io.download to use data.desi.lbl.gov (PR #972).
- Use middle of exposure for barycentric correction time (PR #971).

1.7.50 0.34.4 (2020-04-21)

- Add *desi_proc –batch-opts ...* option for specifying extras like queue reservation (direct push to master).

1.7.51 0.34.3 (2020-04-17)

- Run desi_proc arc and flat jobs on max 10 nodes instead of 5 (PR #958).

1.7.52 0.34.2 (2020-04-16)

- Include *data/spec-arc-lamps.dat* with installed data.
- Mask high readnoise CCD amps (PR #957).

1.7.53 0.34.1 (2020-04-15)

- Expanded scan range for y traceshifts from +-3 to +-10 A (commit 26279d8 direct to master)
- Improved traceshift robusteness for very large shifts of arcs (PR *#954*).
- Added scripts for creating bad pixels masks from darks (PR #946).
- etc/desispec.module use desi_spectro_calib tag 0.1.1 (PR #955).
- import specter only if needed to run, not requiring it just to import desispec.io (PR #955).

Note: *python setup.py install* of this version incorrectly doesn't copy *data/spec-arc-lamps.dat* into the final installed data directory; that is fixed in next version, and was fixed by hand in NERSC 0.34.1 install.

1.7.54 0.34.0 (2020-04-13)

Compatibility notes:

- Requires desiutil >= 2.0.3 (PR #951).
- Backwards incompatible change to sky model format (PR #939.

Changes:

- Refactor S/N fit for QA (PR #917)
- Speed up QA (PR #917)

- Don't mask extreme mask fiberflat >2 or <0.1 in routine autocalib_fiberflat because the fiberflat includes the throughput difference between spectrographs (push to master to address issue #897).
- Modify overscan methods. Default is to no longer analyze the ORSEC region (PR #838).
- Fix sky subtraction with ivar=0 (PR #920).
- Tweaks for logging nightly redshifts and srun (PR #921).
- Added calib config management utilities (PR #926).
- Coadd robustness when missing a camera (PR #927).
- Shorter desi_proc job names (PR #928).
- Set fiberstatus to mask fibers in bad regions of CCDs (PR #930).
- Fix code generating fits reserved keyword warnings (PR #933, #935).
- Try fibermap header if primary header doesn't have RA,DEC (PR #934).
- Force assemble_fibermap for nights before or during 20200310 (PR #936).
- Don't fit traceshifts in y for dome and twilight flats (PR #937).
- Calculate sky model throughput corrections when making sky model instead of while applying model. Note: changes data model. (PR #939).
- Improve averaging of fiberflats (PR #940).
- Fix incorrect multiple calls to bary_corr depending upon MPI parallelism, and merge extract main and main_mpi (PR #943).
- Propagate MJD to spectra fibermap (PR #944).
- Generate spectra files by default and don't coadd across cameras (PR #945).
- Allow coadding across cameras of coadds (PR #948).
- Implement fibermaps per camera (PR #949).
- Use desiutil.iers.freeze_iers instead of desisurvey; requires desiutil>=2.0.3 (PR #951).
- Module file users desi_spectro_calib tag 0.1

1.7.55 0.33.0 (2020-03-05)

- Metadata bookkeeping for early CMX data (PR #857)
- Improved PSF handling in desi_proc (PR #858)
- Modeling scattered light (PR #859, #861, #862)
- desi_proc –calibnight option (PR #860)
- expanding flux calib stdstar bits (PR #862)
- new assemble_fibermap script (PR #864, #902)
- improved sky subtraction and flux calibration robustness (PR #865)
- new desi_group_tileframes script; coadd frames directly (PR #866)
- flux calibration improvements (PR #868, #871, #880, #898)
- more efficient desi_proc –batch parallelism packing (PR #869)
- new desi_proc_dashboard script (PR #870, #901)

- new desi_dailyproc script (PR #872, #881, #895)
- more robustness to missing inputs (PR #875, #876, #883)
- groundwork for improving cosmics masking (PR #878)
- enable barycentric correction in desi_proc (PR #879)
- new plot_spectra script (PR #890)
- new desi_nightly_redshifts script (PR #892)
- Generate QA for a given night + QA bug fixes (PR #894)
- coadd metadata propagation (PR #900)
- don't use FIBERSTATUS!=0 spectra in coadds (PR #903)
- desi_proc more control options for minisv2 run (PR #904)
- Two hotfixes to master to re-enable daily processing:
  - make assemble_fibermap more robust to missing input columns in the platmaker coordinates files.
  - better packing of extraction MPI ranks

1.7.56 0.32.1 (2019-12-27)

- Integration test simulate past not current date to workaound pixsim header mismatch with `DESI_SPECTRO_CALIB` calibrations. (direct push to master).

1.7.57 0.32.0 (2019-12-22)

- Adding more desi_proc options (PR #848, #850).
- Support PSF bootstrapping with broken fibers (PR #849).
- Hot fixes to desi_proc crashes (pushed directly to master).
- Increase cframe task from 1 min to 2 min (direct to master).
- Adapt to new spectrograph SMn naming (PR #853).
- Workaround fitsio bug by setting blank keywords to `None`; adapt to new fiberassign file names (PR #855).

1.7.58 0.31.0 (2019-10-31)

First CMX release with bug fixes for on-sky data.

- Use rrdesi –no-mpi-abort feature (PR #823).
- Added code to generate pixflats (PR #824).
- Support extractions of data without fibermaps (PR #825).
- Propagate FIBERMAP into preproc files (not just frames) (PR #825 and #829).
- Allow extraction wavelenghts slightly off CCD (PR #836).
- PSF I/O pause before merging (PR #836).
- Add *bin/desi_proc* single-exposure processing script (PR #837).
- Use OBSTYPE instead of FLAVOR for desi_qproc (PR #839).

- Bug fix for desi_proc double application of fiberflat (PR #841).
- desi_proc options for non-default PSF and fiberflat (PR #842).
- Correct fibermap to match what petal we are in (PR #843).
- Update database loading to match current data model (PR #844).
- Added desi_proc –batch option (PR #845).

1.7.59 0.30.0 (2019-10-17)

- qproc updates (PR #787).
- QL bias (PR #789).
- Heliocentric corrections (PR #790).
- Update photometric filter usages (PR #791).
- Add gain output option to desi_compute_gain
- Modify overscan subtraction algorithm in desi.preproc.preproc (PR #793).
- Cleanup timing parameters (PR #794).
- Pipeline docs (PR #797).
- Correct for dark trail in raw images (PR #798).
- *yaml.load()* to *yaml.save_load()* (PR #801).
- help numba know the types (PR #802).
- desi_pipe getready fix (PR #803).
- Move raw data transfer scripts to desitransfer (PR #804).
- spectra coaddition (PR #805).
- memory constraints and load balancing (PR #806 and #809).
- preproc header keywords CCDSEC1-4 vs. A-D (PR #807).
- Add *desi_pipe status* command (PR #810).
- Convert any expid input into an int in QA (PR #814).
- Support new FIBERASSIGN_X/Y instead of DESIGN_X/Y (PR #821).
- Added hostname and jobid to task logging (PR #822).

1.7.60 0.29.0 (2019-05-30)

- Add HPSS backup to the raw data transfer script (PR #765).
- Update `desispec.database.redshift` for latest changes in fiberassign tile file data model (PR #770).
- Constants, docs, and test cleanup (PR #771, #773, #776).
- Tune cosmics masking parameters (PR #775).
- Add desi_compute_pixmask (PR #777).
- qproc updates for more flexibility and exposure flavors (PR #778).
- Better io.findfile camera checks (PR #780).

- Support SV1_DESI_TARGET (PR #786).

1.7.61 0.28.0 (2019-02-28)

- Update (non-essential) transfer script for spectrograph functional verification tests (PR #758).
- New calibration data access (inc var. DESI_SPECTRO_CALIB replacing DESI_CCD_CALIBRATION_DATA) (PR #753).
- Fix offline QA S/N vs. mag fits (PR #763).

1.7.62 0.27.1 (2019-01-28)

- QL updates for January 2019 readiness review (PRs #750, #751, #752, #754, #755, #756, #757).

1.7.63 0.27.0 (2018-12-16)

- DB loading targets columns *PRIORITY_INIT* and *NUMOBS_INIT*; requires desitarget/0.27.0 or later for DB loading (PR #747).
- Fix S/N QA when inputs have NaNs (PR #746).
- DB exposures table loading allows NaN entries for RA,DEC,SEEING,etc. for arc and flat calib exposures (PR #743).
- Use new *desiutil.dust.ext_odonnell* function during flux-calibration (PR #736).
- Add support for average flux calibration model in ccd_calibration_data repo (PR #735).
- Support mockobs fibermap format with fewer columns (PR #733).
- Upgrade data transfer script and add additional scripts (PR #732).
- Fix desi_zcatalog RA_TARGET vs. TARGET_RA (PR #723).
- Update redshift database data model and workaround a minor bad data problem (PR #722).
- Refactor offline QA (S/N) to work with updated object typing
- Drop *contam_target* DB truth column; no longer in truth files (one-line commit to master, no PR).
- Bug fix in QA (S/N) + refactor exposure slurping (PR #746)
- Refactor QA_Exposures, QA_Night, and QA_Prod; Generate new Prod QA (offline)

1.7.64 0.26.0 (2018-11-08)

Major non-backwards compatible changes:

- Update to new fibermap format for consistency with targeting and fiber assignment (PR #717).
- Include GAIN in preproc headers (PR #715).
- Prototype data transfer status report webpage (PR #714).
- Integrate qproc/qframe into quicklook (PR #713).
- Quicklook flux calib and config edits (PR #707).

1.7.65 0.25.0 (2018-10-24)

- QL algorithm, config, and format updates (PRs #699, #701, #702). (Includes non-backwards compatible changes).

1.7.66 0.24.0 (2018-01-05)

- Quicklook updates (including non-backwards compatible changes)
  - New QL calibration QA metrics (PR #677).
  - Update QL to use xytraceset instead of custom PSF (PR #682).
  - Cleanup for robustness and maintainability (PR #693).
- Offline QA updates
  - Integrates QL S/N QA into offline QA Frame object (PR #675).
  - Additional offline QA plots on S/N (PR #691).
- Spectroscopic pipeline updates
  - Option to generate bash scripts instead of slurm scripts (PR #686).
  - new *desi_pipe go –resume* option (PR #687).
  - *desi_pipe sync –force-spec-done* option (PR #692)
- Miscellaneous
  - Work-around bug that forbids opening memory-mapped files in update mode on some NERSC filesystems (PR #689).
  - Do not compress image masks (PR #696).
  - Ensure that FITS files specify FITS-standard-compliant units (PR #673).
  - Integration test fixes (PR #695).

1.7.67 0.23.1 (2018-08-09)

- Support STD/STD_FSTAR/STD_FAINT bit names (PR #674).

1.7.68 0.23.0 (2018-07-26)

- Adds qproc algorithms and QFrame class (PR #664).
- Adds *desi_pipe go* for production running (PR #666).
- Increase job maxtime for edison realtime queue (PR #667).
- Updates for running desispec on BOSS data (PR #669).
- Fix QL for list vs. array change in specter/master (PR #670).

1.7.69 0.22.1 (2018-07-18)

- Update processing of QL metrics (PR #659).
- Refactor pipeline and integration test (PR #660).
- Update redshift database to handle changes to fiberassign data model (PR #662).
- Allow rows to be filtered when loading the redshift database (PR #663).

1.7.70 0.22.0 (2018-06-30)

This is the version used for mock observing in June 2018. It includes an update to the directory substructure where raw data are found, grouping each exposure into a separate directory *$DESI_SPECTRO_DATA/{YEARMMDD}/{EXPID}/*.

- Faster traceshift code; requires numba (PR #634).
- Fixed integration tests (PR #635).
- Give empty HDUs am `EXTNAME` (PR #636).
- Update redshift database loading in integration test (PR #638).
- Integration test DB loading (PR #640).
- Move ccd_calibration.yaml to SVN repo (PR #641).
- Logging QA metric status for QLF (PR #642).
- Supporting both new and old fibermap via io.read_fibermap (PP #643).
- Faster lower memory preproc using numba (PR #644)
- ivar bugfix in resample_flux interpolation (PR #646).
- Many QL updates from mock observing (PR #648).
- Raw data in NIGHT/EXPID/. instead of NIGHT/. (PR #648).
- Fix cosmics masking near masked saturated pixels (PR #649).
- Update edison realtime queue config to 25 nodes (PR #650).
- trace_shift code supports PSF formats without "PSF" HDU (PR #654).
- Change keyword `clobber` to `overwrite` in functions from `astropy.io.fits` (PR #658).

1.7.71 0.21.0 (2018-05-25)

Major updates including non-backwards compatible changes to QL output format and pipeline updates for semi-realtime nightly processing.

- Pipeline fix to allow redrock to use a full node per healpix (PR #585).
- Update pipeline maxtime/maxnodes job calculation (PR #588).
- Better sync of pixel tasks and DB sync bugfixes (PR #590).
- Improved handling of errors in case of full job failure (PR #592).
- QA speedups and improvements (PR #593)
  - Add ability to load Frame without reading Resolution matrix
  - Refactor offline QA to use qaprod_dir more sensibly

  - Include hooks in QA to previous fiberflat file location (calib2d)
  - Inhibit scatter plot in skyredidual QA
- Pass MAG into output zbest file (PR #595)
- Allow running multiple task types in a single job (PR #601).
- Pipeline hooks for processing a single exposure (PR #604).
- Override PSF file psferr to avoid masking bright lines. Requires specter > 0.8.1 (PR #606).
- QL QA reorganization (PR #577, #600, #607, #613).
- Integration test and QA fixes (PR #602 and #605).
- New desi_night scripts for semi-realtime processing (PR #609).
- Spectro teststand calibration/utility code updates (PR #610)
- QL S/N vs. mag updates (PR #611)
- QL resampling fixes (PR #615)
- Merge database modules (PR #616).
- Add flexure tests to QL (PR #617).
- Added cori and edison realtime queue support (PR #618, #619, #624).
- QL output format updates (PR #623).

1.7.72 0.20.0 (2018-03-29)

Multiple non-backwards compatible changes:

- Astropy 2 compatibility (PR #519).
- Update Travis tests to recent versions.
- Integration test fixes (PR #552).
- Adds pipeline db count_task_states (PR #552).
- Standardize spectro filenames/locations (PR #545 and #559).
- Complete rewrite of task pipelining (PR #520, #523, #536, #537, #538, #540, #543, #544, #547,)
- QL format updates (#517, #554)
- module file set DESI_CCD_CALIBRATION_DATA (#564).
- Optionally include RA,DEC in merged zcatalog (#562).
- QL updates to S/N calculations (#556).
- fix BUNIT, HPXNSIDE, HPXPIXEL keywords (PR #566)

1.7.73 0.19.0 (2018-03-01)

- Update DB loading for desitarget 0.19.0 targets; make DB loading API less specific to datachallenge directory structure (PR #516).

1.7.74 0.18.0 (2018-02-23)

- Replace deprecated scipy.stats.chisqprob with scipy.stats.distributions.chi2.sf for compatibility with scipy 1.0. (PR #503)
- Faster desi_group_spectra that also propagates SCORES table (PR #505 and #507)
- Add options for fitting spatially non-uniform sky (PR #506)
- Fix logger redirection (PR #508)
- Add hooks for MPI extraction timing benchmarks (PR #509)
- QuickLook metric renaming (PR #512)

1.7.75 0.17.2 (2018-01-30)

- Trace shift optimizations from analyzing teststand data (PR #482).
- Minor QA edits to accommodate minitest (PR #489)
- Additional QA edits including qaprod_root() method (PR #490)
- Introduce QA_Night, QA_MultiExp and refactor QA_Prod accordingly (PR #491)
- Add SCORES HDU to frame files (PR #492)

1.7.76 0.17.1 (2017-12-20)

- Refactors spectral regouping to be faster and derive fibermap format from inputs (PR #473).
- Removed deprecated Brick class, and unused coadds and redmonder zfind that were using Bricks (PR #473).
- Adds skyline QA; fixes QA version usage (PR #458).
- Fixes write_bintable bug if extname=None; fixes missing header comments
- spectro DB database loading updates (PR #477).
- trace shift updates for fiber flats (PR #479).
- Pipeline scaling updates (PR #459 and #466).

1.7.77 0.17.0 (2017-11-10)

- Enabled specter.extract.ex2d nsubbundles option for faster extractions. Requires specter 0.8.1 (PR #451).
- Fixed bug in `desispec.parallel.dist_discrete()` (PR #446)
- Tuned pipeline for scaling tests (PR #457)
- Improved wavelength fitting (via specex update) and sky model error propagation (PR #459)
- Added QL fiberflat, py3 fixes, updated algorithms and config
- Many other QL updates (PR #462)
- Enables MPI parallelism for desi_extract_spectra script (PR #448)

1.7.78 0.16.0 (2017-09-29)

- Small fixes to desi_qa_prod and qa_prod
- Removes a number of QL metrics from offline qa
- Fixes integration tests for desisim newexp refactor
- Removes spectra grouping by brick; nside=64 healpix grouping default
- Add get_nights method to io.meta (PR #422)
- Add search_for_framefile method to io.frame (PR #422)
- Add desi_qa_frame script to generate frame QA (PR #424)
- Add frame_meta to parameters (for slurping the Frame headers) (PR #425)
- Add get_reduced_frames() method to io.meta (PR #425)
- Modifies QA_Prod meta file output to be JSON (PR #425)
- Add load_meta() method to QA_Exposure (PR #425)
- Add time_series ploting to desi_qa_prod (PR #425)
- Add several new plots for skysub residuals (PR #425)
- Adds method to generate QA Table for Prod (PR #425)
- Refactor of skysubresid script (PR #425)
- Refactor QA files to sit in their own folder tree (PR #429)
- Generate HTML files with links to QA figures (PR #429)
- Enable generation of Exposure level QA (PR #429)
- Normalize fiberflat QA by fiber area (PR #429)
- Fixed exptime in fluxcalib ZP calculation (PR #429)
- Added find_exposure_night() method (PR #429)
- Add MED_SKY metric to QA and bright/dark flag in desi_qa_prod
- Update pipeline code for specex and redrock (PR #439 and #440)
- Adds code for adjusting trace locations to match sky lines (PR #433)
- Updates to DB loading (PR #431)
- Adds pixelflat code (PR #426)

1.7.79 0.15.2 (2017-07-12)

- Make the loading of libspecex through ctypes more robust and portable.
- QL configuration cleanup (PR #389).
- Add extrapolate option to resample_flux (PR #415).
- Sphinx and travis tests fixes.

1.7.80 0.15.1 (2017-06-19)

- Fixed `desispec.io.findfile()` path for zbest and coadd (PR #411).
- Add Notebook tutorial: introduction to reading and manipulating DESI spectra (PR #408, #410).
- Update quicklook configuration (PR #395).
- Rename `Spectra.fmap` attribute to `Spectra.fibermap` (PR #407).
- Enable `desi_group_spectra` to run without pipeline infrastructure (PR #405).
- Update desispec.io.findfile spectra path to match dc17a (PR #404).
- Load redshift catalog data from healpix-based zbest files (PR #402).

1.7.81 0.15.0 (2017-06-15)

- Refactor database subpackage and enable loading of both quicksurvey and pipeline outputs (PR #400).
- Clean up pipeline script naming to be grouped by night.
- Modify pipeline to use Spectra objects grouped by HEALPix pixels instead of bricks. Add entry point to regroup cframe data by pixel (PR #394).
- Add a new class, Spectra, which encapsulates a grouping of 1D spectra in one or more bands. Includes selection, updating, and I/O.
- Removed `desispec.brick` as it's now in `desiutil.brick` (PR #392).
- Added function to calculate brick vertices at a given location (PR #388).
- Added function to calculate brick areas at a given location (PR #384).
- Add scripts for submitting nightly job chains.
- Production creation now correctly handles slicing by spectrograph.
- Pipeline job concurrency now computed based on task run time and efficient packing.
- Set default brick size to 0.25 sq. deg. in desispec.brick (PR #378).
- Added function to calculate BRICKID at a given location (PR #378).
- Additional LOCATION, DEVICE_LOC, and PETAL_LOC columns for fibermap (PR #379).
- Create util.py in tests/ which is intended to contain methods to facilitate test runs
- Add vette() method for Frame class (PR #386)
- Began a desispec parameter file: data/params/desispec_param.yml
- Flux calibration improvements (PR #390).

1.7.82 0.14.0 (2017-04-13)

- Replace all instances of *`desispec.log`* with `desiutil.log`; *`get_logger()`* now prints a warning that users need to switch.
- Working DTS delivery script and DTS simulator (PR #367).
- Preproc updates for crosstalk and teststand data (PR #370).
- Flux calibration algorithm updates (PR #371).

- Adds quicklook integration test (PR #361).
- Fixes brickname calculation (PR #373).

1.7.83 0.13.2 (2017-03-27)

- Add framework for DTS delivery and nightly processing scripts (PR #365).
- Force documentation errors to cause Travis errors (PR #364).

1.7.84 0.13.1 (2017-03-03)

- Fix installation of `data/ccd/ccd_calibration.yaml`.

1.7.85 0.13.0 (2017-03-03)

- Fix brick update corruption (PR #314).
- Close PSF file after initializing PSF object.
- Refactor `desispec.io.database` to use SQLAlchemy.
- Fix `graph_path()` usage in workers.
- Update *`desispec.io.raw.write_raw()`* to enable writing simulated raw data with new headers.
- Allow `test_bootcalib` to run even if NERSC portal is returning 403 errors.
- Add `bricksize` property to desispec.brick.Bricks; allow *desispec.brick.Bricks.brickname* to specify bricksize.
- Do SVD inverses when cholesky decompositions fail in fiberflat, sky subtraction, and flux calibration.
- Algorithm updates for teststand and BOSS data
- pipeline updates for docker/shifter
- quicklook updates

1.7.86 0.12.0 (2016-11-09)

- Update integration test to use stdstar_templates_v1.1.fits.
- Support asymmetric resolution matrices (PR #288).
- Quicklook updates (PR #294, #293, #285).
- Fix BUNIT and wavelength f4 *versus* f8.
- Significant pipeline code refactor (PR #300 and #290).
- fix docstrings for sphinx build (PR #308).

1.7.87 0.11.0 (2016-10-14)

- Update template Module file to reflect DESI+Anaconda infrastructure.
- Update redmonster wrapper for reproducibility.
- *desispec.io.brick.BrickBase.get_target_ids* returns target IDs in the order they appear in input file.
- Set BUNIT header keywords (PR #284).
- Improved pipeline logging robustness.
- MPI updates for robustness and non-NERSC operation.
- More py3 fixes.

1.7.88 0.10.0 (2016-09-10)

PR #266 update for Python 3.5:

- Many little updates to work for both python 2.7 and 3.5.
- Internally fibermap is now a `Table` instead of `FITS_rec` table.
- Bug fix for flux calibration QA.
- Requires desiutil >= 1.8.0.

1.7.89 0.9.0 (2016-08-18)

PR #258 (requires specter >= 0.6.0)

- Propagate pixel model goodness of fit to flag outliers from unmasked cosmics.
- desi_extract_spectra –model option to output 2D pixel model
- fix pipeline bug in call to desi_bootcalib (no –qafig option)
- adds extraction tests

Misc:

- desi_qa_skysub – plots residuals (PR #259)
- More quicklook QA (PR #260 and #262)
- Added support for template groups in redmonster (PR #255)
- Lots more pipeline docs (PR #261)

1.7.90 0.8.1 (2016-07-18)

- added QA_Prod
- refactor of fluxcalib QA
- fixed pipeline QA figure output (pdf vs. yaml)

1.7.91 0.8.0 (2016-07-14)

- bootcalib robustness improvements
- improved fibermap propagation
- PRODNAME -> SPECPROD, TYPE -> SPECTYPE
- meaningful batch job names for each step
- better test coverage; more robust to test data download failures
- more quicklook metrics
- used for "oak1" production

1.7.92 0.7.0 and prior

- No changes.rst yet

1.8 Full desispec API Reference

1.8.1 desispec

Tools for DESI spectroscopic processing.

1.8.2 desispec.bootcalib

Utility functions to perform a quick calibration of DESI data

TODO: 1. Expand to r, i cameras 2. QA plots 3. Test with CR data

desispec.bootcalib.**bootcalib**(*deg*, *flatimage*, *arcimage*)

Parameters

- **deg** – Legendre polynomial degree to use to fit
- **flatimage** – desispec.image.Image object of flatfield
- **arcimage** – desispec.image.Image object of arc

Mostly inherited from desispec/bin/desi_bootcalib directly as needed

Returns xfit, fdicts, gauss, all_wave_soln

TODO: document what those return objects are

desispec.bootcalib.**extract_sngfibers_gaussianpsf**(*img*, *img_ivar*, *xtrc*, *sigma*, *box_radius=2*, *verbose=True*)

Extract spectrum for fibers one-by-one using a Gaussian PSF

Parameters

- **img** (*ndarray*) – Image
- **img_ivar** (*ndarray*) – Image inverse variance
- **xtrc** (*ndarray*) – fiber trace
- **sigma** (*float*) – Gaussian sigma for PSF

- **box_radius** (*int, optional*) – Radius for extraction (+/-)

Returns **spec** – Extracted spectrum

Return type ndarray

desispec.bootcalib.**fiber_gauss_new**(*flat*, *xtrc*, *xerr*, *box_radius=2*, *max_iter=5*, *debug=False*, *verbose=False*)

Find the PSF sigma for each fiber This serves as an initial guess to what follows

Parameters

- **flat** (*ndarray of fiber flat image*) –
- **xtrc** (*ndarray of fiber traces*) –
- **xerr** (*ndarray of error in fiber traces*) –
- **box_radius** (*int, optinal*) – Radius of boxcar extraction in pixels
- **max_iter** (*int, optional*) – Maximum number of iterations for rejection

Returns list of Gaussian sigma

Return type gauss

desispec.bootcalib.**fiber_gauss_old**(*flat*, *xtrc*, *xerr*, *box_radius=2*, *max_iter=5*, *debug=False*, *verbose=False*)

Find the PSF sigma for each fiber This serves as an initial guess to what follows

Parameters

- **flat** (*ndarray of fiber flat image*) –
- **xtrc** (*ndarray of fiber traces*) –
- **xerr** (*ndarray of error in fiber traces*) –
- **box_radius** (*int, optinal*) – Radius of boxcar extraction in pixels
- **max_iter** (*int, optional*) – Maximum number of iterations for rejection

Returns list of Gaussian sigma

Return type gauss

desispec.bootcalib.**find_arc_lines**(*spec*, *rms_thresh=7.0*, *nwidth=5*)

Find and centroid arc lines in an input spectrum

Parameters

- **spec** (*ndarray*) – Arc line spectrum
- **rms_thresh** (*float*) – RMS threshold scale
- **nwidth** (*int*) – Line width to test over

desispec.bootcalib.**find_fiber_peaks**(*flat*, *ypos=None*, *nwidth=5*, *debug=False*, *thresh=None*)

Find the peaks of the fiber flat spectra Preforms book-keeping error checking

Parameters

- **flat** – ndarray of fiber flat image
- **ypos** – int [optional] Row for finding peaks Default is half-way up the image
- **nwidth** – int [optional] Width of peak (end-to-end)
- **debug** – bool, optional

Returns

xpk, ypos, cut list of xpk (nearest pixel) at ypos ndarray of cut through the image

desispec.bootcalib.**fit_traces**(*xset*, *xerr*, *func='legendre'*, *order=6*, *sigrej=20.0*, *RMS_TOLER=0.03*, *verbose=False*)

Fit the traces Default is 6th order Legendre polynomials

Parameters

- **xset** (*ndarray*) – traces
- **xerr** (*ndarray*) – Error in the trace values (999.=Bad)
- **RMS_TOLER** (*float, optional [0.02]*) – Tolerance on size of RMS in fit

Returns

- *xnew, fits*
- **xnew** (*ndarray*) – New fit values (without error)
- **fits** (*list*) – List of the fit dicts

desispec.bootcalib.**fix_ycoeff_outliers**(*xcoeff*, *ycoeff*, *deg=5*, *tolerance=2*)

Fix outliers in coefficients for wavelength solution, assuming a continuous function of CCD coordinates

Parameters

- **ncoeff]** (*ycoeff[nfiber,*) – 2D array of Legendre coefficients for X(wavelength)
- **ncoeff]** – 2D array of Legendre coefficients for Y(wavelength)

Options: deg : integer degree of polynomial to fit tolerance : replace fibers with difference of wavelength solution larger than this number of pixels after interpolation

Returns new_ycoeff[nfiber, ncoeff] with outliers replaced by interpolations

For each coefficient, fit a polynomial vs. fiber number with one pass of sigma clipping. Remaining outliers are than replaced with the interpolated fit value.

desispec.bootcalib.**id_arc_lines_using_triplets**(*id_dict*, *w*, *dwdy_prior*, *d2wdy2_prior=1.5e-05*, *toler=0.2*, *ntrack=50*, *nmax=40*)

Match (as best possible), a set of the input list of expected arc lines to the detected list

Parameters

- **id_dict** (*dictionnary with Pixel locations of detected arc lines in "pixpk" and fluxes in "flux"*) –
- **w** (*ndarray*) – array of expected arc lines to be detected and identified
- **dwdy** (*float*) – Average dispersion in the spectrum
- **d2wdy2_prior** (*float*) – Prior on second derivative
- **toler** (*float, optional*) – Tolerance for matching (20%)
- **ntrack** (*max. number of solutions to be tracked*) –

Returns **id_dict** – dict of identified lines

Return type dict

desispec.bootcalib.**load_arcline_list**(*camera*, *vacuum=True*, *lamps=None*)
Loads arc line list from NIST files Parses and rejects

Taken from PYPIT

Parameters

- **lines** (*list*) – List of ions to load
- **vacuum** (*bool, optional*) – Use vacuum wavelengths
- **lamps** (*optional numpy array of ions, ex np.array(["HgI", "CdI","ArI","NeI"])*) –

Returns **alist** – Table of arc lines

Return type Table

desispec.bootcalib.**load_gdarc_lines**(*camera*, *llist*, *vacuum=True*, *lamps=None*, *good_lines_filename=None*)
Loads a select set of arc lines for initial calibrating

Parameters

- **camera** (*str*) – Camera ('b', 'g', 'r')
- **llist** (*table of lines to use, with columns Ion, wave*) –
- **vacuum** (*bool, optional*) – Use vacuum wavelengths
- **lamps** (*optional numpy array of ions, ex np.array(["HgI", "CdI","ArI","NeI"])*) –

Returns

- **dlamb** (*float*) – Dispersion for input camera
- **wmark** (*float*) – wavelength to key off of [???]
- **gd_lines** (*ndarray*) – Array of lines expected to be recorded and good for ID
- **line_guess** (*int or None*) – Guess at the line index corresponding to wmark (default is to guess the 1/2 way point)

desispec.bootcalib.**load_parse_dict**()
Dicts for parsing Arc line lists from NIST

Rejected lines are in the rejected_lines.yaml file

desispec.bootcalib.**parse_nist**(*ion*, *vacuum=True*)
Parse a NIST ASCII table.

Note that the long —- should have been commented out and also the few lines at the start.

Taken from PYPIT

Parameters

- **ion** (*str*) – Name of ion
- **vaccuum** (*bool, optional*) – Use vacuum wavelengths

desispec.bootcalib.**parse_nist_tbl**(*tbl*, *parse_dict*)
Parses a NIST table using various criteria

Parameters

- **tbl** (*Table*) – Read previously from NIST ASCII file

- **parse_dict** (*dict*) – Dict of parsing criteria. Read from load_parse_dict

Returns **tbl** – Rows meeting the criteria

Return type Table

desispec.bootcalib.**qa_arc_spec**(*all_spec*, *all_soln*, *pp*, *figsz=None*)
Generate QA plots of the arc spectra with IDs

Parameters

- **all_spec** – Arc 1D fiber spectra
- **all_soln** – Wavelength solutions
- **pp** – PDF file pointer
- **figsz** – figure size, optional

desispec.bootcalib.**qa_fiber_Dx**(*xfit*, *fdicts*, *pp=None*, *figsz=None*)
Show the spread in the trace per fiber

Used to diagnose the traces

Parameters

- **xfit** – traces
- **fdicts** – dict of the traces
- **pp** – PDF file pointer
- **figsz** – figure size, optional

desispec.bootcalib.**qa_fiber_arcrms**(*all_soln*, *pp*, *figsz=None*)
Show the RMS of the wavelength solutions vs. fiber

Parameters

- **all_soln** – Wavelength solutions
- **pp** – PDF file pointer
- **figsz** – figure size, optional

desispec.bootcalib.**qa_fiber_dlamb**(*all_spec*, *all_soln*, *pp*, *figsz=None*)
Show the Dlamb of the wavelength solutions vs. fiber

Parameters

- **all_soln** – Wavelength solutions
- **pp** – PDF file pointer
- **figsz** – figure size, optional

desispec.bootcalib.**qa_fiber_gauss**(*gauss*, *pp=None*, *figsz=None*)
Show the Gaussian (sigma) fits to each fiber

Parameters

- **gauss** – Gaussian of each fiber
- **pp** – PDF file pointer
- **figsz** – figure size, optional

desispec.bootcalib.**qa_fiber_peaks**(*xpk*, *cut*, *pp=None*, *figsz=None*, *nper=100*)
Generate a QA plot for the fiber peaks

Parameters

- **xpk** – x positions on the CCD of the fiber peaks at a ypos
- **cut** – Spatial cut through the detector
- **pp** – PDF file pointer
- **figsz** – figure size, optional
- **nper** – number of fibers per row in the plot, optional

desispec.bootcalib.**qa_fiber_trace**(*flat*, *xtrc*, *outfil=None*, *Nfiber=25*, *isclmin=0.5*)

Generate a QA plot for the fiber traces

Parameters

- **flat** (*ndarray*) – image
- **xtrc** (*ndarray*) – Trace array
- **isclmin** (*float, optional [0.5]*) – Fraction of 90 percentile flux to scale image by
- **outfil** (*str, optional*) – Output file
- **normalize** (*bool, optional*) – Normalize the flat? If not, use zscale for output

desispec.bootcalib.**reject_lines**(*tbl*, *rej_dict*, *rej_tol=0.1*)

Rejects lines from a NIST table

Taken from PYPIT

Parameters

- **tbl** (*Table*) – Read previously from NIST ASCII file
- **rej_dict** (*dict*) – Dict of rejected lines
- **rej_tol** (*float, optional*) – Tolerance for matching a line to reject to linelist (Angstroms)

Returns **tbl** – Rows not rejected

Return type Table

desispec.bootcalib.**script_bootcalib**(*arc_idx*, *flat_idx*, *cameras=None*, *channels=None*, *nproc=10*)

Runs desi_bootcalib on a series of preproc files

Returns script_bootcalib([0,1,2,3,4,5,6,7,8,9], [10,11,12,13,14])

desispec.bootcalib.**trace_crude_init**(*image*, *xinit0*, *ypass*, *invvar=None*, *radius=2.0*, *maxshift0=0.5*, *maxshift=0.15*, *maxerr=0.2*)

Python port of trace_crude_idl.pro from IDLUTILS

Modified for initial guess

Parameters

- **image** (*2D ndarray*) – Image for tracing
- **xinit** (*ndarray*) – Initial guesses for trace peak at ypass
- **ypass** (*int*) – Row for initial guesses

Returns

- **xset** (*Trace for each fiber*)

- **xerr** (*Estimated error in that trace*)

desispec.bootcalib.**trace_fweight**(*fimage*, *xinit*, *ycen=None*, *invvar=None*, *radius=2.0*, *debug=False*)
Python port of trace_fweight.pro from IDLUTILS

Parameters

- **fimage** (*2D ndarray*) – Image for tracing
- **xinit** (*ndarray*) – Initial guesses for x-trace
- **invvar** (*ndarray, optional*) – Inverse variance array for the image
- **radius** (*float, optional*) – Radius for centroiding; default to 3.0

desispec.bootcalib.**write_psf**(*outfile*, *xfit*, *fdicts*, *gauss*, *wv_solns*, *legendre_deg=5*, *without_arc=False*, *XCOEFF=None*, *fiberflat_header=None*, *arc_header=None*, *fix_ycoeff=True*)
Write the output to a Base PSF format

Parameters

- **outfile** (*str*) – Output file
- **xfit** (*ndarray*) – Traces
- **gauss** (*list*) – List of gaussian sigmas
- **fdicts** (*list*) – List of trace fits
- **wv_solns** (*list*) – List of wavelength calibrations
- **ncoeff** (*int*) – Number of Legendre coefficients in fits

1.8.3 desispec.calibfinder

Reading and selecting calibration data from $DESI_SPECTRO_CALIB using content of image headers

desispec.calibfinder.**_load_smsp**()
Loads $DESI_SPECTRO_CALIB/spec/smsp.txt into global _sp2sm and _sm2sp dicts

desispec.calibfinder.**badfibers**(*headers, keys=['BROKENFIBERS', 'BADCOLUMNFIBERS', 'LOWTRANSMISSIONFIBERS'], yaml_file=None*)
find list of bad fibers from $DESI_SPECTRO_CALIB using the keywords found in the headers

Parameters **headers** – list of fits headers, or list of dictionnaries

Optional: keys: list of keywords, among ["BROKENFIBERS","BADCOLUMNFIBERS","LOWTRANSMISSIONFIBERS"]. Default is all of them. yaml_file: path to a specific yaml file. By default, the code will automatically find the yaml file from the environment variable DESI_SPECTRO_CALIB and the CAMERA keyword in the headers

Returns List of bad fibers as a 1D array of intergers

desispec.calibfinder.**findcalibfile**(*headers*, *key*, *yaml_file=None*)
read and select calibration data file from $DESI_SPECTRO_CALIB using the keywords found in the headers

Parameters

- **headers** – list of fits headers, or list of dictionnaries
- **key** – type of calib file, e.g. 'PSF' or 'FIBERFLAT'

Optional: yaml_file: path to a specific yaml file. By default, the code will automatically find the yaml file from the environment variable DESI_SPECTRO_CALIB and the CAMERA keyword in the headers

Returns path to calibration file

desispec.calibfinder.**parse_date_obs**(*value*)
converts DATE-OBS keywork to int with for instance DATE-OBS=2016-12-21T18:06:21.268371-05:00

desispec.calibfinder.**sm2sp**(*sm*, *night=None*)
Converts spectrograph sm hardware number to sp logical number

Parameters **sm** – spectrograph sm number 1-10 or str sm[1-10]

Returns "spP" if input is str "smM", or int P if input is int M

Note: uses $DESI_SPECTRO_CALIB/spec/smsp.txt

TODO: add support for different mappings based on night

desispec.calibfinder.**sp2sm**(*sp*)
Converts spectrograph sp logical number to sm hardware number

Parameters **sp** – spectrograph int 0-9 or str sp[0-9]

Returns "smM" if input is str "spP", or int M if input is int P

Note: uses $DESI_SPECTRO_CALIB/spec/smsp.txt

TODO: add support for different mappings based on night

Coadd spectra

desispec.coaddition.**coadd**(*spectra*, *cosmics_nsig=0.0*, *onetile=False*)
Coadd spectra for each target and each camera, modifying input spectra obj.

Parameters **spectra** – desispec.spectra.Spectra object

Options: cosmics_nsig: float, nsigma clipping threshold for cosmics rays onetile: bool, if True, inputs are from a single tile

Notes: if *onetile* is True, additional tile-specific columns like LOCATION and FIBER are included the FIBERMAP; otherwise these are only in the EXP_FIBERMAP since for the same target they could be different on different tiles.

desispec.coaddition.**coadd_cameras**(*spectra*, *cosmics_nsig=0.0*, *onetile=False*)
Return coadd across both exposures and cameras

Parameters **spectra** – desispec.spectra.Spectra object

Options: cosmics_nsig: float, nsigma clipping threshold for cosmics rays onetile: bool, if True, inputs are from a single tile

If *onetile* is True, additional tile-specific columns like LOCATION and FIBER are included the FIBERMAP; otherwise these are only in the EXP_FIBERMAP since for the same target they could be different on different tiles.

Note: unlike *coadd*, this does not modify the input spectra object

desispec.coaddition.**coadd_fibermap**(*fibermap*, *onetile=False*)
Coadds fibermap

Parameters **fibermap** (*Table or ndarray*) – fibermap of individual exposures

Options: onetile (bool): this is a coadd of a single tile, not across tiles

Returns: (coadded_fibermap, exp_fibermap) Tables

coadded_fibermap contains the coadded_fibermap for the columns that can be coadded, while exp_fibermap is the subset of columns of the original fibermap that can't be meaningfully coadded because they are per-exposure quantities like FIBER_X.

If onetile is True, the coadded_fibermap includes additional columns like MEAN_FIBER_X that are meaningful if coadding a single tile, but not if coadding across tiles.

desispec.coaddition.**decorrelate_divide_and_conquer**(*Cinv*, *Cinvf*, *wavebin*, *flux*, *ivar*, *rdata*)

Decorrelate an inverse covariance using the matrix square root.

Implements the decorrelation part of the spectroperfectionism algorithm described in Bolton & Schlegel 2009 (BS) http://arxiv.org/abs/0911.2689.

with the divide and conquer approach, i.e. per diagonal block of the matrix, with an overlapping 'skin' from one block to another.

Parameters

- **Cinv** – Square 2D array: input inverse covariance matrix
- **Cinvf** – 1D array: input
- **wavebin** – minimal size of wavelength bin in A, used to define the core and skin size
- **flux** – 1D array: output flux (has to be allocated)
- **ivar** – 1D array: output flux inverse variance (has to be allocated)
- **rdata** – 2D array: output resolution matrix per diagonal (has to be allocated)

desispec.coaddition.**fast_resample_spectra**(*spectra*, *wave*)

Fast resampling of spectra file. The output resolution = Id. The neighboring flux bins are correlated.

Parameters

- **spectra** – desispec.spectra.Spectra object
- **wave** – 1D numy array with new wavelenght grid

Returns desispec.spectra.Spectra object, resolution data=Id

desispec.coaddition.**get_resampling_matrix**(*global_grid*, *local_grid*, *sparse=False*)

Build the rectangular matrix that linearly resamples from the global grid to a local grid.

The local grid range must be contained within the global grid range.

Parameters

- **global_grid** (*numpy.ndarray*) – Sorted array of n global grid wavelengths.
- **local_grid** (*numpy.ndarray*) – Sorted array of m local grid wavelengths.

Returns Array of (m,n) matrix elements that perform the linear resampling.

Return type numpy.ndarray

desispec.coaddition.**resample_spectra_lin_or_log**(*spectra*, *linear_step=0*, *log10_step=0*, *fast=False*, *wave_min=None*, *wave_max=None*, *nproc=1*)

Resampling of spectra file.

Parameters

- **spectra** – desispec.spectra.Spectra object
- **linear_step** – if not null the ouput wavelenght grid will be linear with this step
- **log10_step** – if not null the ouput wavelenght grid will be logarthmic with this step

Options: fast: simple resampling. fast but at the price of correlated output flux bins and no information on resolution wave_min: if set, use this min wavelength wave_max: if set, use this max wavelength

Returns desispec.spectra.Spectra object

desispec.coaddition.**spectroperf_resample_spectra**(*spectra*, *wave*, *nproc=1*)

Resampling of spectra file using the spectrophotometic approach

Parameters

- **spectra** – desispec.spectra.Spectra object
- **wave** – 1D numy array with new wavelenght grid

Returns desispec.spectra.Spectra object

1.8.4 desispec.cosmics

Utility functions to find cosmic rays

desispec.cosmics.**_reject_cosmic_rays_ala_sdss_single**(*pix*, *ivar*, *selection*, *psf_gradients*, *nsig*, *cfudge*, *c2fudge*)

Cosmic ray rejection following the implementation in SDSS/BOSS. (see idlutils/src/image/reject_cr_psf.c and idlutils/pro/image/reject_cr.pro)

This routine is a single call, similar to IDL routine reject_cr_single Called by reject_cosmic_rays_ala_sdss with an iteration

Input is a pre-processed image : desispec.Image Ouput is a rejection mask of the same size as the image

A faster version of this routine using numba in implemented in _reject_cosmic_rays_ala_sdss_single_numba

Parameters

- **pix** – input desispec.Image.pix (counts in pixels, 2D image)
- **ivar** – inverse variance of pix
- **selection** – array of booleans
- **psf_gradients** – 1D array of size 4, for 4 axes: horizontal,vertical and 2 diagonals
- **nsig** – number of sigma above background required
- **cfudge** – number of sigma inconsistent with PSF required
- **c2fudge** – fudge factor applied to PSF

desispec.cosmics.**_reject_cosmic_rays_ala_sdss_single_numba**(*pix*, *ivar*, *selection*, *psf_gradients*, *nsig*, *cfudge*, *c2fudge*)

Cosmic ray rejection following the implementation in SDSS/BOSS. (see idlutils/src/image/reject_cr_psf.c and idlutils/pro/image/reject_cr.pro)

This routine is a single call, similar to IDL routine reject_cr_single Called by reject_cosmic_rays_ala_sdss with an iteration

Input is a pre-processed image : desispec.Image Ouput is a rejection mask of the same size as the image

This routine is much faster than _reject_cosmic_rays_ala_sdss_single (if you have numba installed, otherwise it is catastrophically slower)

Parameters

- **pix** – input desispec.Image.pix (counts in pixels, 2D image)
- **ivar** – inverse variance of pix
- **selection** – array of booleans
- **psf_gradients** – 1D array of size 4, for 4 axes: horizontal,vertical and 2 diagonals
- **nsig** – number of sigma above background required
- **cfudge** – number of sigma inconsistent with PSF required
- **c2fudge** – fudge factor applied to PSF

desispec.cosmics.**reject_cosmic_rays**(*img*, *nsig=5.0*, *cfudge=3.0*, *c2fudge=0.9*, *niter=100*, *dilate=True*)

Cosmic ray rejection Input is a pre-processed image : desispec.Image The image mask is modified

Parameters **img** – input desispec.Image

desispec.cosmics.**reject_cosmic_rays_1d**(*frame*, *nsig=3*, *psferr=0.05*)

Use resolution matrix in frame to detect spikes in the spectra that are narrower than the PSF, and mask them

desispec.cosmics.**reject_cosmic_rays_ala_sdss**(*img*, *nsig=6.0*, *cfudge=3.0*, *c2fudge=0.5*, *niter=6*, *dilate=True*)

Cosmic ray rejection following the implementation in SDSS/BOSS. (see idlutils/src/image/reject_cr_psf.c and idlutils/pro/image/reject_cr.pro)

This routine is calling several times reject_cosmic_rays_ala_sdss_single, similar to IDL routine reject_cr. There is an optionnal dilatation of the mask by one pixel, as done in sdssproc.pro for SDSS

Input is a pre-processed image : desispec.Image Ouput is a rejection mask of the same size as the image

Parameters

- **img** – input desispec.Image
- **nsig** – number of sigma above background required
- **cfudge** – number of sigma inconsistent with PSF required
- **c2fudge** – fudge factor applied to PSF
- **niter** – number of iterations on neighboring pixels of rejected pixels
- **dilate** – force +1 pixel dilation of rejection mask

1.8.5 desispec.database

Tools for loading DESI data into databases.

At the present time, this covers:

1. Raw metadata database.
2. Redshift results database (a.k.a "SpectraDB").

It does *not* include:

1. Pipeline processing status database.

2. Imaging and targeting databases.

1.8.6 desispec.database.metadata

Code for interacting with the file metadatabase.

class desispec.database.metadata.**Brick**(***kwargs*)
Representation of a region of the sky.

class desispec.database.metadata.**BrickStatus**(***kwargs*)
Representation of the status of a particular *Brick*.

class desispec.database.metadata.**ExposureFlavor**(***kwargs*)
List of exposure flavors. Used to constrain the possible values.

class desispec.database.metadata.**Frame**(***kwargs*)
Representation of a particular pointing, exposure, spectrograph and band (a.k.a. 'channel' or 'arm').

class desispec.database.metadata.**FrameStatus**(***kwargs*)
Representation of the status of a particular *Frame*.

class desispec.database.metadata.**Night**(***kwargs*)
List of observation nights. Used to constrain the possible values.

class desispec.database.metadata.**Status**(***kwargs*)
List of possible processing statuses.

class desispec.database.metadata.**Tile**(**args*, ***kwargs*)
Representation of a particular pointing of the telescope.

_coarse_overlapping_bricks(*session*)
Get the bricks that *may* overlap a tile.

Parameters **session** (sqlalchemy.orm.session.Session) – Database connection.

Returns A list of *Brick* objects.

Return type list

_constants()
Define mathematical constants associated with a tile.

area
Area of the tile in steradians.

brick_offset(*brick*)
Offset a brick in the same way as a tile.

Parameters **brick** (*Brick*) – A brick.

Returns A tuple containing the shifted ra1 and ra2.

Return type tuple()

circum_square
Defines a square-like region on the sphere which circumscribes the tile.

cos_radius
Cosine of the radius, precomputed for speed.

offset(*shift=10.0*)
Provide an offset to move RA away from wrap-around.

Parameters **shift** (float, optional) – Amount to offset in degrees.

Returns An amount to offset in degrees.

Return type float

overlapping_bricks(*session*, *map_petals=False*)
Perform a geometric calculation to find bricks that overlap a tile.

Parameters

- **session** (sqlalchemy.orm.session.Session) – Database connection.
- **map_petals** (*bool, optional*) – If True a map of petal number to a list of overlapping bricks is returned.

Returns If *map_petals* is False, a list of Polygon objects. Otherwise, a dict mapping petal number to the *Brick* objects that overlap that petal.

Return type list

petals(*Npetals=10*)
Convert a tile into a set of Wedge objects.

Parameters **Npetals** (int, optional) – Number of petals (default 10).

Returns A list of Wedge objects.

Return type list

radius
Radius of tile in degrees.

simulate_frame(*session*, *band*, *spectrograph*, *flavor='science'*, *exptime=1000.0*)
Simulate a DESI frame given a Tile object.

Parameters

- **session** (sqlalchemy.orm.session.Session) – Database connection.
- **band** (str) – 'b', 'r', 'z'
- **spectrograph** (int) – Spectrograph number [0-9].
- **flavor** (str, optional) – Exposure flavor (default 'science').
- **exptime** (float, optional) – Exposure time in seconds (default 1000).

Returns A tuple containing a *Frame* object ready for loading, and a list of bricks that overlap.

Return type tuple

desispec.database.metadata.**get_all_tiles**(*session*, *obs_pass=0*, *limit=0*)
Get all tiles from the database.

Parameters

- **session** (sqlalchemy.orm.session.Session) – Database connection.
- **obs_pass** (int, optional) – Select only tiles from this pass.
- **limit** (int, optional) – Limit the number of tiles returned

Returns A list of Tiles.

Return type list

desispec.database.metadata.**load_data**(*session*, *datapath*)
Load a night or multiple nights into the frame table.

Parameters

- **session** (sqlalchemy.orm.session.Session) – Database connection.
- **datapath** (str) – Name of a data directory.

Returns A list of the exposure numbers found.

Return type list

desispec.database.metadata.**load_simulated_data**(*session*, *obs_pass=0*)
Load simulated frame and brick data.

Parameters

- **session** (sqlalchemy.orm.session.Session) – Database connection.
- **obs_pass** (int, optional) – If set, only simulate one pass.

desispec.database.metadata.**main**()
Entry point for command-line script.

Returns An integer suitable for passing to sys.exit().

Return type int

1.8.7 desispec.database.redshift

Code for loading spectroscopic pipeline results (specifically redshifts) into a database.

class desispec.database.redshift.**FiberAssign**(***kwargs*)
Representation of the fiberassign table.

class desispec.database.redshift.**ObsList**(***kwargs*)
Representation of the obslist table.

class desispec.database.redshift.**SchemaMixin**
Mixin class to allow schema name to be changed at runtime. Also automatically sets the table name.

class desispec.database.redshift.**Target**(***kwargs*)
Representation of the target table.

class desispec.database.redshift.**Truth**(***kwargs*)
Representation of the truth table.

class desispec.database.redshift.**ZCat**(***kwargs*)
Representation of the zcat table.

desispec.database.redshift.**get_options**(**args*)
Parse command-line options.

Parameters **args** (*iterable*) – If arguments are passed, use them instead of sys.argv.

Returns The parsed options.

Return type argparse.Namespace

desispec.database.redshift.**load_fiberassign**(*datapath*, *maxpass=4*, *hdu='FIBERASSIGN'*, *q3c=False*, *latest_epoch=False*, *last_column='NUMOBS_MORE'*)
Load fiber assignment files into the fiberassign table.

Tile files can appear in multiple epochs, so for a given tileid, load the tile file with the largest value of epoch. In the "real world", a tile file appears in each epoch until it is observed, therefore the tile file corresponding to the actual observation is the one with the largest epoch.

Parameters

- **datapath** (str) – Full path to the directory containing tile files.
- **maxpass** (int, optional) – Search for pass numbers up to this value (default 4).
- **hdu** (int or str, optional) – Read a data table from this HDU (default 'FIBERASSIGN').
- **q3c** (bool, optional) – If set, create q3c index on the table.
- **latest_epoch** (bool, optional) – If set, search for the latest tile file among several epochs.
- **last_column** (str, optional) – Do not load columns past this name (default 'NUMOBS_MORE').

desispec.database.redshift.**load_file**(*filepath*, *tcls*, *hdu=1*, *expand=None*, *convert=None*, *index=None*, *rowfilter=None*, *q3c=False*, *chunksize=50000*, *maxrows=0*)

Load a data file into the database, assuming that column names map to database column names with no surprises.

Parameters

- **filepath** (str) – Full path to the data file.
- **tcls** (sqlalchemy.ext.declarative.api.DeclarativeMeta) – The table to load, represented by its class.
- **hdu** (int or str, optional) – Read a data table from this HDU (default 1).
- **expand** (dict, optional) – If set, map FITS column names to one or more alternative column names.
- **convert** (dict, optional) – If set, convert the data for a named (database) column using the supplied function.
- **index** (str, optional) – If set, add a column that just counts the number of rows.
- **rowfilter** (*callable, optional*) – If set, apply this filter to the rows to be loaded. The function should return bool, with True meaning a good row.
- **q3c** (bool, optional) – If set, create q3c index on the table.
- **chunksize** (int, optional) – If set, load database *chunksize* rows at a time (default 50000).
- **maxrows** (int, optional) – If set, stop loading after *maxrows* are loaded. Alteratively, set *maxrows* to zero (0) to load all rows.

desispec.database.redshift.**load_redrock**(*datapath=None*, *hdu='REDSHIFTS'*, *q3c=False*)

Load redrock files into the zcat table.

This function is deprecated since there should now be a single redshift catalog file.

Parameters

- **datapath** (str) – Full path to the directory containing redrock files.
- **hdu** (int or str, optional) – Read a data table from this HDU (default 'REDSHIFTS').
- **q3c** (bool, optional) – If set, create q3c index on the table.

desispec.database.redshift.**main**()

Entry point for command-line script.

Returns An integer suitable for passing to sys.exit().

Return type int

desispec.database.redshift.**q3c_index**(*table*, *ra='ra'*)
Create a q3c index on a table.

Parameters

- **table** (str) – Name of the table to index.
- **ra** (str, optional) – If the RA, Dec columns are called something besides "ra" and "dec", set its name. For example, `ra='target_ra'`.

desispec.database.redshift.**setup_db**(*options=None*, ***kwargs*)
Initialize the database connection.

Parameters

- **options** (argpare.Namespace) – Parsed command-line options.
- **kwargs** (*keywords*) – If present, use these instead of *options*. This is more user-friendly than setting up a `Namespace` object in, *e.g.* a Jupyter Notebook.

Returns `True` if the configured database is a PostgreSQL database.

Return type bool

desispec.database.redshift.**update_truth**(*filepath*, *hdu=2*, *chunksize=50000*, *skip=('SLOPES', 'EMLINES')*)
Add data from columns in other HDUs of the Truth table.

Parameters

- **filepath** (str) – Full path to the data file.
- **hdu** (int or str, optional) – Read a data table from this HDU (default 2).
- **chunksize** (int, optional) – If set, update database *chunksize* rows at a time (default 50000).
- **skip** (tuple(), optional) – Do not load columns with these names (default, `('SLOPES', 'EMLINES')`)

1.8.8 desispec.database.util

Classes and functions for use by all database code.

desispec.database.util.**convert_dateobs**(*timestamp*, *tzinfo=None*)
Convert a string *timestamp* into a datetime.datetime object.

Parameters

- **timestamp** (str) – Timestamp in string format.
- **tzinfo** (datetime.tzinfo, optional) – If set, add time zone to the timestamp.

Returns The converted *timestamp*.

Return type datetime.datetime

desispec.database.util.**parse_pgpass**(*hostname='nerscdb03.nersc.gov'*, *username='desidev_admin'*)
Read a `~/.pgpass` file.

Parameters

- **hostname** (str, optional) – Database hostname.
- **username** (str, optional) – Database username.

Returns A string suitable for creating a SQLAlchemy database engine, or None if no matching data was found.

Return type str

1.8.9 desispec.fiberflat

Utility functions to compute a fiber flat correction and apply it We try to keep all the (fits) io separated.

desispec.fiberflat.**apply_fiberflat**(*frame*, *fiberflat*)

Apply fiberflat to frame. Modifies frame.flux and frame.ivar. Checks whether an heliocentric correction has been applied to the frame wavelength in which case also apply it to the flat field array.

Parameters

- **frame** – *desispec.Frame* object
- **fiberflat** – *desispec.FiberFlat* object

The frame is divided by the fiberflat, except where the fiberflat=0.

frame.mask gets bit specmask.BADFIBERFLAT set where

- fiberflat.fiberflat == 0
- fiberflat.ivar == 0
- fiberflat.mask != 0

desispec.fiberflat.**autocalib_fiberflat**(*fiberflats*)

Combine fiberflats of all spectrographs from different lamps to maximize uniformity :param fiberflats: list of *desispec.FiberFlat* object

returns a dictionary of desispec.FiberFlat objects , one per spectrograph

desispec.fiberflat.**average_fiberflat**(*fiberflats*)

Average several fiberflats :param fiberflats: list of *desispec.FiberFlat* object

returns a desispec.FiberFlat object

desispec.fiberflat.**compute_fiberflat**(*frame*, *nsig_clipping=10.0*, *accuracy=0.0005*, *minval=0.1*, *maxval=10.0*, *max_iterations=15*, *smoothing_res=5.0*, *max_bad=100*, *max_rej_it=5*, *min_sn=0*, *diag_epsilon=0.001*)

Compute fiber flat by deriving an average spectrum and dividing all fiber data by this average. Input data are expected to be on the same wavelength grid, with uncorrelated noise. They however do not have exactly the same resolution.

Parameters

- **frame** (*desispec.Frame*) – input Frame object with attributes wave, flux, ivar, resolution_data
- **nsig_clipping** – [optional] sigma clipping value for outlier rejection
- **accuracy** – [optional] accuracy of fiberflat (end test for the iterative loop)
- **minval** – [optional] mask pixels with flux < minval * median fiberflat.
- **maxval** – [optional] mask pixels with flux > maxval * median fiberflat.
- **max_iterations** – [optional] maximum number of iterations

- **smoothing_res** – [optional] spacing between spline fit nodes for smoothing the fiberflat
- **max_bad** – [optional] mask entire fiber if more than max_bad-1 initially unmasked pixels are masked during the iterations
- **max_rej_it** – [optional] reject at most the max_rej_it worst pixels in each iteration
- **min_sn** – [optional] mask portions with signal to noise less than min_sn
- **diag_epsilon** – [optional] size of the regularization term in the deconvolution

Returns

desispec.FiberFlat object with attributes wave, fiberflat, ivar, mask, meanspec

Notes: - we first iteratively :

- compute a deconvolved mean spectrum
- compute a fiber flat using the resolution convolved mean spectrum for each fiber
- smooth the fiber flat along wavelength
- clip outliers
- then we compute a fiberflat at the native fiber resolution (not smoothed)
- the routine returns the fiberflat, its inverse variance , mask, and the deconvolved mean spectrum
- the fiberflat is the ratio data/mean , so this flat should be divided to the data

NOTE THAT THIS CODE HAS NOT BEEN TESTED WITH ACTUAL FIBER TRANSMISSION VARIATIONS, OUTLIER PIXELS, DEAD COLUMNS ...

desispec.fiberflat.**qa_fiberflat**(*param*, *frame*, *fiberflat*)

Calculate QA on FiberFlat object

Parameters

- **param** – dict of QA parameters
- **frame** – Frame
- **fiberflat** – FiberFlat

Returns

dict of QA outputs Need to record simple Python objects for yaml (str, float, int)

Return type qadict

1.8.10 desispec.fluxcalibration

Flux calibration routines.

desispec.fluxcalibration.**ZP_from_calib**(*exptime*, *wave*, *calib*)

Calculate the ZP in AB magnitudes given the calibration and the wavelength arrays :param exptime: float; exposure time in seconds :param wave: 1D array (A) :param calib: 1D array (converts erg/s/A to photons/s/A)

Returns 1D array of ZP values in AB magnitudes

Return type ZP_AB

desispec.fluxcalibration.**_compute_coef**(*coord*, *node_coords*)

Function used by interpolate_on_parameter_grid2

Parameters

- **coord** – 1D array of coordinates of size n_axis
- **node_coords** – 2D array of coordinates of nodes, shape = (n_nodes,n_axis)

Returns 1D array of linear coefficients for each node, size = n_nodes

Return type coef

desispec.fluxcalibration.**_func**(*arg*)
Used for multiprocessing.Pool

desispec.fluxcalibration.**_func2**(*arg*)
Used for multiprocessing.Pool

desispec.fluxcalibration.**_smooth_template**(*template_id*, *camera_index*, *template_flux*)
Used for multiprocessing.Pool

desispec.fluxcalibration.**applySmoothingFilter**(*flux*, *width=200*)
Return a smoothed version of the input flux array using a median filter

Parameters

- **flux** – 1D array of flux
- **width** – size of the median filter box

Returns median filtered flux of same size as input

Return type smooth_flux

desispec.fluxcalibration.**apply_flux_calibration**(*frame*, *fluxcalib*)
Applies flux calibration to input flux and ivar

Parameters

- **frame** – Spectra object with attributes wave, flux, ivar, resolution_data
- **fluxcalib** – FluxCalib object with wave, calib, ...

Modifies frame.flux and frame.ivar

desispec.fluxcalibration.**compute_flux_calibration**(*frame*, *input_model_wave*, *input_model_flux*, *input_model_fibers*, *nsig_clipping=10.0*, *deg=2*, *debug=False*, *highest_throughput_nstars=0*, *exposure_seeing_fwhm=1.1*, *stdcheck=True*, *nsig_flux_scale=3*)
Compute average frame throughput based on data frame.(wave,flux,ivar,resolution_data) and spectrophotometrically calibrated stellar models (model_wave,model_flux). Wave and model_wave are not necessarily on the same grid

Parameters

- **frame** – Frame object with attributes wave, flux, ivar, resolution_data
- **input_model_wave** – 1D[nwave] array of model wavelengths
- **input_model_flux** – 2D[nstd, nwave] array of model fluxes
- **input_model_fibers** – 1D[nstd] array of model fibers
- **nsig_clipping** – (optional) sigma clipping level

- **exposure_seeing_fwhm** – (optional) seeing FWHM in arcsec of the exposure
- **stdcheck** – check if the model stars are actually standards according to the fibermap and only rely on those
- **nsig_flux_scale** – n sigma cutoff on the flux scale among standard stars

Returns desispec.FluxCalib object calibration: mean calibration (without resolution)

Notes

- we first resample the model on the input flux wave grid
- then convolve it to the data resolution (the input wave grid is supposed finer than the spectral resolution)
- then iteratively - fit the mean throughput (deconvolved, this is needed because of sharp atmospheric absorption lines) - compute a scale factor to fibers (to correct for small mis-alignement for instance) - perform outlier rejection

There is one subtelty with the relation between calibration and resolution.

- The input frame flux is on average flux^frame_fiber = R_fiber*C*flux^true where C is the true calibration (or throughput) which is a function of wavelength. This is the system we solve.
- But we want to return a calibration vector per fiber C_fiber defined by flux^cframe_fiber = flux^frame_fiber/C_fiber, such that flux^cframe can be compared with a convolved model of the truth, flux^cframe_fiber = R_fiber*flux^true, i.e. (R_fiber*C*flux^true)/C_fiber = R_fiber*true_flux, giving C_fiber = (R_fiber*C*flux^true)/(R_fiber*flux^true)
- There is no solution for this for all possible input specta. The solution for a flat spectrum is returned, which is very close to C_fiber = R_fiber*C (but not exactly).

desispec.fluxcalibration.**interpolate_on_parameter_grid**(*data_wave*, *data_flux*, *data_ivar*, *template_flux*, *teff*, *logg*, *feh*, *template_chi2*)

3D Interpolation routine among templates based on a grid of parameters teff, logg, feh. The tricky part is to define a cube on the parameter grid populated with templates, and it is not always possible. The routine never extrapolates, so that we stay in the range of input parameters.

Parameters

- **data_wave** – 1D[nwave] array of wavelength (concatenated list of input wavelength of different cameras and exposures)
- **data_flux** – 1D[nwave] array of normalized flux = (input flux)/median_filter(input flux) (concatenated list)
- **data_ivar** – 1D[nwave] array of inverse variance of normalized flux
- **template_flux** – 2D[ntemplates,nwave] array of normalized flux of templates (after resample, convolution and division by median_filter)
- **teff** – 1D[ntemplates]
- **logg** – 1D[ntemplates]
- **feh** – 1D[ntemplates]

- **template_chi2** – 1D[ntemplatess] array of precomputed chi2 = sum(data_ivar*(data_flux-template_flux)**2)

Returns best fit coefficient of linear combination of templates chi2 : chi2 of the linear combination

Return type coefficients

desispec.fluxcalibration.**isStdStar**(*fibermap*, *bright=None*)

Determines if target(s) are standard stars

Parameters **fibermap** – table including DESI_TARGET or SV1_DESI_TARGET bit mask(s)

Optional: bright: if True, only bright time standards; if False, only darktime, otherwise both

Returns bool or array of bool

desispec.fluxcalibration.**match_templates**(*wave*, *flux*, *ivar*, *resolution_data*, *stdwave*, *stdflux*, *teff*, *logg*, *feh*, *ncpu=1*, *z_max=0.005*, *z_res=2e-05*, *template_error=0*, *comm=None*)

For each input spectrum, identify which standard star template is the closest match, factoring out broadband throughput/calibration differences.

Parameters

- **wave** – A dictionary of 1D array of vacuum wavelengths [Angstroms]. Example below.
- **flux** – A dictionary of 1D observed flux for the star
- **ivar** – A dictionary 1D inverse variance of flux
- **resolution_data** – resolution corresponding to the star's fiber
- **stdwave** – 1D standard star template wavelengths [Angstroms]
- **stdflux** – 2D[nstd, nwave] template flux
- **teff** – 1D[nstd] effective model temperature
- **logg** – 1D[nstd] model surface gravity
- **feh** – 1D[nstd] model metallicity
- **ncpu** – number of cpu for multiprocessing
- **comm** – MPI communicator; if given, ncpu will be ignored and only rank 0 will return results that are not None

Returns numpy.array of linear coefficient of standard stars redshift : redshift of standard star chipdf : reduced chi2

Return type coef

Notes

- wave and stdwave can be on different grids that don't necessarily overlap
- wave does not have to be uniform or monotonic. Multiple cameras can be supported by concatenating their wave and flux arrays

desispec.fluxcalibration.**normalize_templates**(*stdwave*, *stdflux*, *mag*, *band*, *photsys*)

Returns spectra normalized to input magnitudes.

Parameters

- **stdwave** – 1D array of standard star wavelengths [Angstroms]
- **stdflux** – 1D observed flux
- **mag** – float desired magnitude
- **band** – G,R,Z,W1 or W2
- **photsys** – N or S (for Legacy Survey North or South)

Returns same as input normflux : normalized flux array

Return type stdwave

Only SDSS_r band is assumed to be used for normalization for now.

desispec.fluxcalibration.**qa_fluxcalib**(*param*, *frame*, *fluxcalib*)

Parameters

- **param** – dict of QA parameters
- **frame** – Frame
- **fluxcalib** – FluxCalib

Returns

dict of QA outputs Need to record simple Python objects for yaml (str, float, int)

Return type qadict

desispec.fluxcalibration.**redshift_fit**(*wave*, *flux*, *ivar*, *resolution_data*, *stdwave*, *stdflux*, *z_max=0.005*, *z_res=5e-05*, *template_error=0.0*)

Redshift fit of a single template

Parameters

- **wave** – A dictionary of 1D array of vacuum wavelengths [Angstroms]. Example below.
- **flux** – A dictionary of 1D observed flux for the star
- **ivar** – A dictionary 1D inverse variance of flux
- **resolution_data** – resolution corresponding to the star's fiber
- **stdwave** – 1D standard star template wavelengths [Angstroms]
- **stdflux** – 1D[nwave] template flux
- **z_max** – float, maximum blueshift and redshift in scan, has to be positive
- **z_res** – float, step of of redshift scan between [-z_max,+z_max]
- **template_error** – float, assumed template flux relative error

Returns redshift of standard star

Return type redshift

Notes

- wave and stdwave can be on different grids that don't necessarily overlap
- wave does not have to be uniform or monotonic. Multiple cameras can be supported by concatenating their wave and flux arrays

desispec.fluxcalibration.**resample_template**(*data_wave_per_camera*, *resolution_data_per_camera*, *template_wave*, *template_flux*, *template_id*)

Resample a spectral template on the data wavelength grid. Then convolve the spectra by the resolution for each camera. Also returns the result of applySmoothingFilter. This routine is used internally in a call to multiprocessing.Pool.

Parameters

- **data_wave_per_camera** – A dictionary of 1D array of vacuum wavelengths [Angstroms], one entry per camera and exposure.
- **resolution_data_per_camera** – A dictionary of resolution corresponding for the fiber, one entry per camera and exposure.
- **template_wave** – 1D array, input spectral template wavelength [Angstroms] (arbitrary spacing).
- **template_flux** – 1D array, input spectral template flux density.
- **template_id** – int, template identification index, used to ensure matching of input/output after a multiprocessing run.

Returns int, template identification index, same as input. output_wave : A dictionary of 1D array of vacuum wavelengths output_flux : A dictionary of 1D array of output template flux output_norm : A dictionary of 1D array of output template smoothed flux

Return type template_id

1.8.11 desispec.frame

Lightweight wrapper class for spectra, to be returned by io.read_frame Lightweight wrapper class for preprocessed image data

1.8.12 desispec.interpolation

Utility functions for interpolation of spectra over different wavelength grids.

desispec.interpolation.**_unweighted_resample**(*output_x*, *input_x*, *input_flux_density*, *extrapolate=False*)

Returns a flux conserving resampling of an input flux density. The total integrated flux is conserved.

Parameters

- **output_x** – SORTED vector, not necessarily linearly spaced
- **input_x** – SORTED vector, not necessarily linearly spaced
- **input_flux_density** – input flux density dflux/dx sampled at x

both must represent the same quantity with the same unit input_flux_density = dflux/dx sampled at input_x

Options:

extrapolate: extrapolate using edge values of input array, default is False, in which case values outside of input array are set to zero

Returns returns output_flux

This interpolation conserves flux such that, on average, output_flux_density = input_flux_density

The input flux density outside of the range defined by the edges of the first and last bins is considered null. The bin size of bin 'i' is given by (x[i+1]-x[i-1])/2 except for the first and last bin where it is (x[1]-x[0]) and (x[-1]-x[-2]) so flux density is zero for x<x[0]-(x[1]-x[0])/2 and x>x[-1]+(x[-1]-x[-2])/2

The input is interpreted as the nodes positions and node values of a piece-wise linear function:

```
y(x) = sum_i y_i * f_i(x)
```

with::

f_i(x) = (x_{i-1}<x<=x_{i})*(x-x_{i-1})/(x_{i}-x_{i-1})

- (x_{i}<x<=x_{i+1})*(x-x_{i+1})/(x_{i}-x_{i+1})

the output value is the average flux density in a bin flux_out(j) = int_{x>(x_{j-1}+x_j)/2}^{x<(x_j+x_{j+1})/2} y(x) dx / 0.5*(x_{j+1}+x_{j-1})

desispec.interpolation.**resample_flux**(*xout*, *x*, *flux*, *ivar=None*, *extrapolate=False*)

Returns a flux conserving resampling of an input flux density. The total integrated flux is conserved.

Parameters

- **xout** (–) – output SORTED vector, not necessarily linearly spaced
- **x** (–) – input SORTED vector, not necessarily linearly spaced
- **flux** (–) – input flux density dflux/dx sampled at x

both x and xout must represent the same quantity with the same unit

Options:

- ivar: weights for flux; default is unweighted resampling
- extrapolate: extrapolate using edge values of input array, default is False, in which case values outside of input array are set to zero.

Setting both ivar and extrapolate raises a ValueError because one cannot assign an ivar outside of the input data range.

Returns if ivar is None, returns outflux if ivar is not None, returns outflux, outivar

This interpolation conserves flux such that, on average, output_flux_density = input_flux_density

The input flux density outside of the range defined by the edges of the first and last bins is considered null. The bin size of bin 'i' is given by (x[i+1]-x[i-1])/2 except for the first and last bin where it is (x[1]-x[0]) and (x[-1]-x[-2]) so flux density is zero for x<x[0]-(x[1]-x[0])/2 and x>x[-1]-(x[-1]-x[-2])/2

The input is interpreted as the nodes positions and node values of a piece-wise linear function:

```
y(x) = sum_i y_i * f_i(x)
```

with:

```
f_i(x) =     (x_{i-1}<x<=x_{i})*(x-x_{i-1})/(x_{i}-x_{i-1})
          + (x_{i}<x<=x_{i+1})*(x-x_{i+1})/(x_{i}-x_{i+1})
```

the output value is the average flux density in a bin:

```
flux_out(j) = int_{x>(x_{j-1}+x_j)/2}^{x<(x_j+x_{j+1})/2} y(x) dx /  0.5*(x_{j+1}
↪+x_{j-1})
```

1.8.13 desispec.io

Tools for data and metadata I/O.

1.8.14 desispec.io.download

Download files from DESI repository.

desispec.io.download.**_auth**(*machine='data.desi.lbl.gov'*)
Get authentication credentials.

desispec.io.download.**_map_download**(*map_tuple*)
Wrapper function to pass to multiprocess.Pool.map().

desispec.io.download.**download**(*filenames*, *single_thread=False*, *workers=None*, *baseurl='https://data.desi.lbl.gov/desi'*)
Download files from the DESI repository.

This function will try to create any directories that don't already exist, using the exact paths specified in *filenames*.

Parameters

- **filenames** – string or list-like object containing filenames.
- **single_thread** – (optional) if `True`, do not use multiprocessing to download files.
- **workers** – (optional) integer indicating the number of worker processes to create.
- **baseurl** – (optional) string containing the URL of the top-level DESI directory.

Returns Full, local path to the file(s) downloaded.

1.8.15 desispec.io.fiberflat

IO routines for fiberflat.

desispec.io.fiberflat.**read_fiberflat**(*filename*)
Read fiberflat from filename

Parameters **filename** (*str*) – Name of fiberflat file, or (night, expid, camera) tuple

Returns

FiberFlat object with attributes fiberflat, ivar, mask, meanspec, wave, header

Notes

fiberflat, ivar, mask are 2D [nspec, nwave] meanspec and wave are 1D [nwave]

desispec.io.fiberflat.**write_fiberflat**(*outfile*, *fiberflat*, *header=None*, *fibermap=None*)
Write fiberflat object to outfile

Parameters

- **outfile** – filepath string or (night, expid, camera) tuple
- **fiberflat** – FiberFlat object

Optional: header: dict or fits.Header object to use as HDU 0 header fibermap: table to store as FIBERMAP HDU

Returns filepath of file that was written

1.8.16 desispec.io.fibermap

IO routines for fibermap.

desispec.io.fibermap.**_set_fibermap_columns**()
Prepare survey-specific list of columns.

Returns As a convenience, return the full set of survey-specific columns.

Return type dict

desispec.io.fibermap.**assemble_fibermap**(*night*, *expid*, *badamps=None*, *badfibers_filename=None*, *force=False*, *allow_svn_override=True*)
Create a fibermap for a given *night* and *expid*.

Parameters

- **night** (int) – YEARMMDD night of sunset.
- **expid** (int) – Exposure ID.
- **badamps** (str, optional) – Comma separated list of "{camera}{petal}{amp}", *i.e.* "[brz][0-9][ABCD]". Example: 'b7D,z8A'.
- **badfibers_filename** (str, optional) – Filename with table of bad fibers with at least two columns: FIBER and FIBERSTATUS
- **force** (bool, optional) – Create fibermap even if missing coordinates/guide files.
- **allow_svn_override** (bool, optional) – If True (default), allow fiberassign SVN to override raw data.

Returns A representation of a fibermap FITS file.

Return type astropy.io.fits.HDUList

desispec.io.fibermap.**compare_fiberassign**(*fa1*, *fa2*, *compare_all=False*)
Check whether two fiberassign tables agree for cols used by ICS/platemaker

Parameters **fa2** (*fa1,*) – fiberassign astropy Tables or numpy structured arrays

Options: compare_all: if True, compare all columns, not just those used by ops

Returns list of columns with mismatches; empty list if all agree

Note: if both are NaN, it is considered a match

desispec.io.fibermap.**empty_fibermap**(*nspec*, *specmin=0*, *survey='main'*)
Return an empty fibermap Table to be filled in.

Parameters

- **nspec** (int) – Number of fibers (spectra) to include.
- **specmin** (int, optional) – Staring spectrum index.
- **survey** (string, optional) – Define columns for this survey; default 'main'.

Returns An empty Table.

Return type Table

desispec.io.fibermap.**fibermap_new2old**(*fibermap*)
Converts new format fibermap into old format fibermap

Parameters **fibermap** – new-format fibermap table (e.g. with FLUX_G column)

Returns old format fibermap (e.g. with MAG column)

Note: this is a transitional convenience function to allow us to simulate new format fibermaps while still running code that expects the old format. After all code has been converted to use the new format, this will be removed.

desispec.io.fibermap.**find_fiberassign_file**(*night*, *expid*, *tileid=None*, *nightdir=None*)
Walk backwards in exposures to find matching fiberassign file

Parameters

- **night** (*int*) – YEARMMDD night of observations
- **expid** (*int*) – spectroscopic exposure ID

Options: tileid (int): tileid to look for nightdir (str): base directory for raw data on that night

Returns first fiberassign file found on or before *expid* on *night*.

Raises FileNotFoundError if no fiberassign file is found

desispec.io.fibermap.**read_fibermap**(*filename*)
Reads a fibermap file and returns its data as an astropy Table

Parameters **filename** – input file name

desispec.io.fibermap.**write_fibermap**(*outfile*, *fibermap*, *header=None*, *clobber=True*, *extname='FIBERMAP'*)
Write fibermap binary table to outfile.

Parameters

- **outfile** (*str*) – output filename
- **fibermap** – astropy Table of fibermap data
- **header** – header data to include in same HDU as fibermap
- **clobber** (*bool, optional*) – overwrite outfile if it exists
- **extname** (*str, optional*) – set the extension name.

Returns full path to filename of fibermap file written.

Return type write_fibermap (str)

desispec.io.filters.**load_filter**(*given_filter*)
Uses speclite.filters to load the filter transmission Returns speclite.filters.FilterResponse object

Parameters

- **given_filter** – given filter for which the qe is to be loaded. Desi templates/
- **have them in uppercase, so it should be in upper case like SDSS, DECAM or**(*files*) –
- **Speclite has lower case so are mapped here.**(*WISE.*) –

desispec.io.filters.**load_gaia_filter**(*band*, *dr=2*)
Uses speclite.filters to load the filter transmission Returns speclite.filters.FilterResponse object

Parameters

- **band** – filter pass-band in "G","BP","RP"

- **dr** – 2 or 3

desispec.io.filters.**load_legacy_survey_filter**(*band*, *photsys*)
Uses speclite.filters to load the filter transmission Returns speclite.filters.FilterResponse object

Parameters

- **band** – filter pass-band in "G","R","Z","W1","W2"
- **photsys** – "N" or "S" for North (BASS+MzLS) or South (CTIO/DECam)

1.8.17 desispec.io.fluxcalibration

IO routines for flux calibration.

desispec.io.fluxcalibration.**read_average_flux_calibration**(*filename*)
Read average flux calibration file; returns an AverageFluxCalib object

desispec.io.fluxcalibration.**read_flux_calibration**(*filename*)
Read flux calibration file; returns a FluxCalib object

desispec.io.fluxcalibration.**read_stdstar_models**(*filename*)
Read stdstar models from filename.

Parameters **filename** (*str*) – File containing standard star models.

Returns flux[nspec, nwave], wave[nwave], fibers[nspec]

Return type read_stdstar_models (tuple)

desispec.io.fluxcalibration.**read_stdstar_templates**(*stellarmodelfile*)
Reads an input stellar model file

Parameters **stellarmodelfile** – input filename

Returns (wave, flux, templateid, teff, logg, feh) tuple: wave : 1D[nwave] array of wavelengths [Angstroms] flux : 2D[nmodel, nwave] array of model fluxes templateid : 1D[nmodel] array of template IDs for each spectrum teff : 1D[nmodel] array of effective temperature for each model logg : 1D[nmodel] array of surface gravity for each model feh : 1D[nmodel] array of metallicity for each model

desispec.io.fluxcalibration.**write_average_flux_calibration**(*outfile*, *averagefluxcalib*)
Writes average flux calibration.

Parameters

- **outfile** – output file name
- **averagefluxcalib** – AverageFluxCalib object

Options: header : dict-like object of key/value pairs to include in header

desispec.io.fluxcalibration.**write_flux_calibration**(*outfile*, *fluxcalib*, *header=None*)
Writes flux calibration.

Parameters

- **outfile** – output file name
- **fluxcalib** – FluxCalib object

Options: header : dict-like object of key/value pairs to include in header

desispec.io.fluxcalibration.**write_stdstar_models**(*norm_modelfile*, *normalizedFlux*, *wave*, *fibers*, *data*, *fibermap*, *input_frames*, *header=None*)

Writes the normalized flux for the best models.

Parameters

- **norm_modelfile** – output file path
- **normalizedFlux** – 2D array of flux[nstdstars, nwave]
- **wave** – 1D array of wavelengths[nwave] in Angstroms
- **fibers** – 1D array of fiberids for these spectra
- **data** – meta data table about which templates best fit
- **fibermap** – fibermaps rows for the input standard stars
- **input_frames** – Table with NIGHT, EXPID, CAMERA of input frames used

1.8.18 desispec.io.frame

I/O routines for Frame objects

desispec.io.frame.**read_frame**(*filename*, *nspec=None*, *skip_resolution=False*)

Reads a frame fits file and returns its data.

Parameters

- **filename** – path to a file, or (night, expid, camera) tuple where night = string YEARMMDD expid = integer exposure ID camera = b0, r1, .. z9
- **skip_resolution** – bool, option Speed up read time (>5x) by avoiding the Resolution matrix

Returns desispec.Frame object with attributes wave, flux, ivar, etc.

desispec.io.frame.**read_meta_frame**(*filename*, *extname=0*)

Load the meta information of a Frame :param filename: path to a file :param extname: int, optional; Extension for grabbing header info

Returns dict or astropy.fits.header

Return type meta

desispec.io.frame.**search_for_framefile**(*frame_file*, *specprod_dir=None*)

Search for an input frame_file in the desispec redux hierarchy :param frame_file: str :param specprod_dir: str, optional

Returns str, full path to frame_file if found else raise error

Return type mfile

desispec.io.frame.**write_frame**(*outfile*, *frame*, *header=None*, *fibermap=None*, *units=None*)

Write a frame fits file and returns path to file written.

Parameters

- **outfile** – full path to output file, or tuple (night, expid, channel)
- **frame** – desispec.frame.Frame object with wave, flux, ivar...

Optional: header: astropy.io.fits.Header or dict to override frame.header fibermap: table to store as FIBERMAP HDU

Returns full filepath of output file that was written

Note: to create a Frame object to pass into write_frame, frame = Frame(wave, flux, ivar, resolution_data)

1.8.19 desispec.io.image

I/O routines for Image objects

desispec.io.image.**read_image**(*filename*)
Returns desispec.image.Image object from input file

desispec.io.image.**write_image**(*outfile*, *image*, *meta=None*)
Writes image object to outfile

Parameters

- **outfile** – output file string
- **image** – desispec.image.Image object (or any object with 2D array attributes image, ivar, mask)

Optional: meta : dict-like object with metadata key/values (e.g. FITS header)

1.8.20 desispec.io.meta

IO metadata functions.

desispec.io.meta.**faflavor2program**(*faflavor*)
Map FAFLAVOR keywords to what we wish we had set for FAPRGRM

Parameters **faflavor** (*str or array of str*) – FAFLAVOR keywords from fiberassign

Returns what FAPRGM would be if we had set it (dark, bright, backup, other)

Return type faprgm (str or array of str)

Note: this was standardized by sv3 and main, but evolved during sv1 and sv2

desispec.io.meta.**find_exposure_night**(*expid*, *specprod_dir=None*)
Find the night that has the exposure :param expid: int :param specprod_dir: str, optional

Returns str

Return type night

desispec.io.meta.**findfile**(*filetype*, *night=None*, *expid=None*, *camera=None*, *tile=None*, *groupname=None*, *nside=64*, *band=None*, *spectrograph=None*, *survey=None*, *faprogram=None*, *rawdata_dir=None*, *specprod_dir=None*, *download=False*, *outdir=None*, *qaprod_dir=None*)
Returns location where file should be

Parameters **filetype** – file type, typically the prefix, e.g. "frame" or "psf"

Args depending upon filetype: night : YEARMMDD string expid : integer exposure id camera : 'b0' 'r1' .. 'z9' tile : integer tile (pointing) number groupname : spectral grouping name (healpix pixel, tile "cumulative" or "pernight") nside : healpix nside band : one of 'b','r','z' identifying the camera band spectrograph

: integer spectrograph number, 0-9 survey : e.g. sv1, sv3, main, special faprogram : fiberassign program, e.g. dark, bright

Options: rawdata_dir : overrides $DESI_SPECTRO_DATA specprod_dir : overrides $DESI_SPECTRO_REDUX/$SPECPROD/ qaprod_dir : defaults to $DESI_SPECTRO_REDUX/$SPECPROD/QA/ if not provided download : if not found locally, try to fetch remotely outdir : use this directory for output instead of canonical location

Raises

- `ValueError` – for invalid file types, and other invalid input
- `KeyError` – for missing environment variables

`desispec.io.meta.`**`get_exposures`**(*night*, *raw=False*, *rawdata_dir=None*, *specprod_dir=None*)

Get a list of available exposures for the specified night.

Exposures are identified as correctly formatted subdirectory names within the night directory, but no checks for valid contents of these exposure subdirectories are performed.

Parameters

- **`night`** (*`str`*) – Date string for the requested night in the format YYYYMMDD.
- **`raw`** (*`bool`*) – Returns raw exposures if set, otherwise returns processed exposures.
- **`rawdata_dir`** (*`str`*) – [optional] overrides $DESI_SPECTRO_DATA
- **`specprod_dir`** (*`str`*) – Path containing the exposures/ directory to use. If the value is None, then the value of *`specprod_root()`* is used instead. Ignored when raw is True.

Returns

List of integer exposure numbers available for the specified night. The list will be empty if no the night directory exists but does not contain any exposures.

Return type list

Raises

- `ValueError` – Badly formatted night date string
- `IOError` – non-existent night.

`desispec.io.meta.`**`get_files`**(*filetype*, *night*, *expid*, *specprod_dir=None*, *qaprod_dir=None*, ***kwargs*)

Get files for a specified exposure.

Uses *`findfile()`* to determine the valid file names for the specified type. Any camera identifiers not matching the regular expression [brz][0-9] will be silently ignored.

Parameters

- **`filetype`** (*`str`*) – Type of files to get. Valid choices are 'frame', 'cframe', 'psf', etc.
- **`night`** (*`str`*) – Date string for the requested night in the format YYYYMMDD.
- **`expid`** (*`int`*) – Exposure number to get files for.
- **`specprod_dir`** (*`str`*) – Path containing the exposures/ directory to use. If the value is None, then the value of *`specprod_root()`* is used instead. Ignored when raw is True.

Returns

Dictionary of found file names using camera id strings as keys, which are guaranteed to match the regular expression [brz][0-9].

Return type dict

desispec.io.meta.**get_nights**(*strip_path=True*, *specprod_dir=None*, *sub_folder='exposures'*)
Generate a list of nights in a given folder (default is exposures/) Demands an 8 digit name beginning with 20

Parameters

- **strip_path** – bool, optional; Strip the path to the nights folders
- **rawdata_dir** –
- **specprod_dir** –
- **sub_root** – str, optional; 'exposures', 'calib2d'

Returns list of nights (without or with paths)

Return type nights

desispec.io.meta.**get_pipe_database**()
Get the production database location based on the environment.

desispec.io.meta.**get_pipe_logdir**()
Return the name of the subdirectory containing pipeline logs.

Returns (str): The name of the subdirectory.

desispec.io.meta.**get_pipe_nightdir**()
Return the name of the subdirectory containing per-night files.

Returns (str): The name of the subdirectory.

desispec.io.meta.**get_pipe_pixeldir**()
Return the name of the subdirectory containing per-pixel files.

Returns (str): The name of the subdirectory.

desispec.io.meta.**get_pipe_rundir**(*specprod_dir=None*)
Return the directory path for pipeline runtime files.

Parameters **specprod_dir** (*str*) – Optional path to production directory. If None, the this is obtained from *specprod_root()*.

Returns (str): the directory path for pipeline runtime files.

desispec.io.meta.**get_pipe_scriptdir**()
Return the name of the subdirectory containing pipeline scripts.

Returns (str): The name of the subdirectory.

desispec.io.meta.**get_raw_files**(*filetype*, *night*, *expid*, *rawdata_dir=None*)
Get files for a specified exposure.

Uses *findfile()* to determine the valid file names for the specified type. Any camera identifiers not matching the regular expression [brz][0-9] will be silently ignored.

Parameters

- **filetype** (*str*) – Type of files to get. Valid choices are 'raw', 'preproc', 'fibermap'.
- **night** (*str*) – Date string for the requested night in the format YYYYMMDD.
- **expid** (*int*) – Exposure number to get files for.
- **rawdata_dir** (*str*) – [optional] overrides $DESI_SPECTRO_DATA

Returns

Dictionary of found file names using camera id strings as keys, which are guaranteed to match the regular expression [brz][0-9].

Return type dict

desispec.io.meta.**get_reduced_frames**(*channels=['b', 'r', 'z'], nights=None, ftype='cframe', **kwargs*)

Loops through a production to find all reduced frames (default is cframes) One can choose a subset of reduced frames by argument :param channels: list, optional :param nights: list, optional :param ftype: str, optional :param kwargs: passed to get_files()

Returns list for frame filenames

Return type all_frames

desispec.io.meta.**qaprod_root**(*specprod_dir=None*)

Return directory root for spectro production QA, i.e. `$DESI_SPECTRO_REDUX/$SPECPROD/QA`.

Raises `KeyError` – if these environment variables aren't set.

desispec.io.meta.**rawdata_root**()

Returns directory root for raw data, i.e. `$DESI_SPECTRO_DATA`

Raises `KeyError` – if these environment variables aren't set.

desispec.io.meta.**shorten_filename**(*filename*)

Attempt to shorten filename to fit in FITS header without CONTINUE

Parameters **filename** (*str*) – input filename

Returns potentially shortened filename

Replaces prefixes from environment variables:

- $DESI_SPECTRO_CALIB -> SPCALIB
- $DESI_SPECTRO_REDUX/$SPECPROD -> SPECPROD

desispec.io.meta.**specprod_root**(*specprod=None*)

Return directory root for spectro production, i.e. `$DESI_SPECTRO_REDUX/$SPECPROD`.

Options: specprod (str): overrides $SPECPROD

Raises `KeyError` – if these environment variables aren't set.

desispec.io.meta.**validate_night**(*night*)

Validates a night string and converts to a date.

Parameters **night** (*str*) – Date string for the requested night in the format YYYYMMDD.

Returns Date object representing this night.

Return type datetime.date

Raises `ValueError` – Badly formatted night string.

1.8.21 desispec.io.params

IO routines for parameter values

desispec.io.params.**read_params**(*filename=None, reload=False*)

Read parameter data from file

1.8.22 desispec.io.qa

IO routines for QA

desispec.io.qa.**load_qa_brick**(*filename*)
Load an existing QA_Brick or generate one, as needed :param filename: str

Returns: qa_brick: QA_Brick object

desispec.io.qa.**load_qa_frame**(*filename*, *frame_meta=None*, *flavor=None*)
Load an existing QA_Frame or generate one, as needed

Parameters

- **filename** – str
- **frame_meta** – dict like, optional
- **flavor** – str, optional Type of QA_Frame

Returns QA_Frame object

Return type qa_frame

desispec.io.qa.**load_qa_multiexp**(*inroot*)
Load QA for a given production

Parameters **inroot** – str base filename without format extension

Returns dict

Return type odict

desispec.io.qa.**qafile_from_framefile**(*frame_file*, *qaprod_dir=None*, *output_dir=None*)
Derive the QA filename from an input frame file :param frame_file: str :param output_dir: str, optional Over-ride default output path :param qa_dir: str, optional Over-ride default QA

Returns:

desispec.io.qa.**read_qa_brick**(*filename*)
Generate a QA_Brick object from a data file

desispec.io.qa.**read_qa_data**(*filename*)
Read data from a QA file

desispec.io.qa.**read_qa_frame**(*filename*)
Generate a QA_Frame object from a data file

desispec.io.qa.**write_qa_brick**(*outfile*, *qabrick*)
Write QA for a given exposure

Parameters

- **outfile** – filename
- **qabrick** – QA_Brick object _data: dict of QA info

desispec.io.qa.**write_qa_exposure**(*outroot*, *qaexp*, *ret_dict=False*)
Write QA for a given exposure

Parameters

- **outroot** – str filename without format extension
- **qa_exp** – QA_Exposure object
- **ret_dict** – bool, optional Return dict only? [for qa_prod, mainly]

Returns str or dict

Return type outfile or odict

desispec.io.qa.**write_qa_frame**(*outfile*, *qaframe*, *verbose=False*)
Write QA for a given frame

Parameters

- **outfile** – str filename
- **qa_exp** – QA_Frame object, with the following attributes qa_data: dict of QA info

desispec.io.qa.**write_qa_multiexp**(*outroot*, *mdict*, *indent=True*)
Write QA for a given production

Parameters

- **outroot** – str filename without format extension
- **mdict** – dict

Returns

str output filename

Return type outfile

desispec.io.qa.**write_qa_ql**(*outfile*, *qaresult*)
Write QL output files

Parameters

- **outfile** – str filename to be written (yaml)
- **qaresult** – dict QAresults from run_qa()

Returns str

Return type outfile

I/O for DESI raw data files

See DESI-1229 for format details TODO: move into datamodel after we have verified the format

desispec.io.raw.**read_raw**(*filename*, *camera*, *fibermapfile=None*, *fill_header=None*, ***kwargs*)
Returns preprocessed raw data from *camera* extension of *filename*.

Parameters

- **filename** (str) – Input FITS filename with DESI raw data.
- **camera** (str) – Camera name (B0, R1, ... Z9) or FITS extension name.
- **fibermapfile** (str, optional) – Read fibermap from this file; if None create blank fibermap.
- **fill_header** (list, optional) – A list of HDU names or numbers. The header cards from these HDUs will be added to the header of the camera HDU read from *filename*.

Returns Image object with member variables pix, ivar, mask, readnoise.

Return type desispec.image.Image

Raises

- IOError – If *camera* is not a HDU in *filename*.

- KeyError – If EXPTIME is not present in any header in *filename*, or if both NIGHT and DATE-OBS are missing from input headers.
- ValueError – If NIGHT in the primary header does not match NIGHT in the camera header, or if *fill_header* is not a list.

Notes

Other keyword arguments are passed to *desispec.preproc.preproc()*, *e.g.* bias, pixflat, mask. See *preproc()* documentation for details.

desispec.io.raw.**write_raw**(*filename*, *rawdata*, *header*, *camera=None*, *primary_header=None*)
Write raw pixel data to a DESI raw data file

Parameters

- **filename** – file name to write data; if this exists, append a new HDU
- **rawdata** – 2D ndarray of raw pixel data including overscans
- **header** – dict-like object or fits.Header with keywords CCDSECx, BIASSECx, DATASECx where x=A,B,C,D

Options: camera : b0, r1 .. z9 - override value in header primary_header : header to write in HDU0 if filename doesn't yet exist

The primary utility of this function over raw fits calls is to ensure that all necessary keywords are present before writing the file. CCDSECx, BIASSECx, DATASECx where x=A,B,C,D DATE-OBS will generate a non-fatal warning if missing

1.8.23 desispec.io.sky

IO routines for sky.

desispec.io.sky.**read_sky**(*filename*)
Read sky model and return SkyModel object with attributes wave, flux, ivar, mask, header.

skymodel.wave is 1D common wavelength grid, the others are 2D[nspec, nwave]

desispec.io.sky.**write_sky**(*outfile*, *skymodel*, *header=None*)
Write sky model.

Parameters

- **outfile** – filename or (night, expid, camera) tuple
- **skymodel** – SkyModel object, with the following attributes wave : 1D wavelength in vacuum Angstroms flux : 2D[nspec, nwave] sky flux ivar : 2D inverse variance of sky flux mask : 2D mask for sky flux stat_ivar : 2D inverse variance of sky flux (statistical only)
- **header** – optional fits header data (fits.Header, dict, or list)

1.8.24 desispec.io.spectra

I/O routines for working with spectral grouping files.

desispec.io.spectra.**read_frame_as_spectra**(*filename*, *night=None*, *expid=None*, *band=None*, *single=False*)

Read a FITS file containing a Frame and return a Spectra.

A Frame file is very close to a Spectra object (by design), and only differs by missing the NIGHT and EXPID in the fibermap, as well as containing only one band of data.

Parameters **infile** (*str*) – path to read

Options: night (int): the night value to use for all rows of the fibermap. expid (int): the expid value to use for all rows of the fibermap. band (str): the name of this band. single (bool): if True, keep spectra as single precision in memory.

Returns (Spectra): The object containing the data read from disk.

desispec.io.spectra.**read_spectra**(*infile*, *single=False*)

Read Spectra object from FITS file.

This reads data written by the write_spectra function. A new Spectra object is instantiated and returned.

Parameters

- **infile** (*str*) – path to read
- **single** (*bool*) – if True, keep spectra as single precision in memory.

Returns (Spectra): The object containing the data read from disk.

desispec.io.spectra.**read_tile_spectra**(*tileid*, *night*, *specprod=None*, *reduxdir=None*, *coadd=False*, *single=False*, *targets=None*, *fibers=None*, *redrock=True*, *group=None*)

Read and return combined spectra for a tile/night

Parameters

- **tileid** (*int*) – Tile ID
- **night** (*int or str*) – YEARMMDD night or tile group, e.g. 'deep' or 'all'

Options: specprod (str) : overrides $SPECPROD reduxdir (str) : overrides $DESI_SPECTRO_REDUX/$SPECPROD coadd (bool) : if True, read coadds instead of per-exp spectra single (bool) : if True, use float32 instead of double precision targets (array-like) : filter by TARGETID fibers (array-like) : filter by FIBER redrock (bool) : if True, also return row-matched redrock redshift catalog group (str) : reads spectra in group (pernight, cumulative, . . .)

Returns: spectra or (spectra, redrock) combined Spectra obj for all matching targets/fibers filter row-matched redrock catalog (if redrock=True)

Raises ValueError if no files or matching spectra are found

Note: the returned spectra are not necessarily in the same order as the *targets* or *fibers* input filters

desispec.io.spectra.**write_spectra**(*outfile*, *spec*, *units=None*)

Write Spectra object to FITS file.

This places the metadata into the header of the (empty) primary HDU. The first extension contains the fibermap, and then HDUs are created for the different data arrays for each band.

Floating point data is converted to 32 bits before writing.

Parameters

- **outfile** (*str*) – path to write
- **spec** (Spectra) – the object containing the data
- **units** (*str*) – optional string to use for the BUNIT key of the flux HDUs for each band.

Returns The absolute path to the file that was written.

1.8.25 desispec.io.util

Utility functions for desispec IO.

desispec.io.util.**_dict2ndarray**(*data*, *columns=None*)
Convert a dictionary of ndarrays into a structured ndarray

Also works if DATA is an AstroPy Table.

Parameters

- **data** – input dictionary, each value is an ndarray
- **columns** – optional list of column names

Returns structured numpy.ndarray with named columns from input data dictionary

Notes

data[key].shape[0] must be the same for every key every entry in columns must be a key of data

Example d = dict(x=np.arange(10), y=np.arange(10)/2) nddata = _dict2ndarray(d, columns=['x', 'y'])

desispec.io.util.**_supports_memmap**(*filename*)
Returns `True` if the filesystem containing *filename* supports opening memory-mapped files in update mode.

desispec.io.util.**add_columns**(*data*, *colnames*, *colvals*)
Adds extra columns to a data table

Parameters

- **data** – astropy Table or numpy structured array
- **colnames** – list of new column names to add
- **colvals** – list of column values to add; each element can be scalar or vector

Returns new table with extra columns added

Example

fibermap = add_columns(fibermap, ['NIGHT', 'EXPID'], [20102020, np.arange(len(fibermap))])

Notes

This is similar to *numpy.lib.recfunctions.append_fields*, but it also accepts astropy Tables as the *data* input, and accepts scalar values to expand as entries in *colvals*.

desispec.io.util.**addkeys**(*hdr1*, *hdr2*, *skipkeys=None*)
Add new header keys from hdr2 to hdr1, skipping skipkeys

Parameters

- **hdr1** (*dict-like*) – destination header for keywords
- **hdr2** (*dict-like*) – source header for keywords

Modifies hdr1 in place

desispec.io.util.**camword_to_spectros**(*camword*, *full_spectros_only=False*)
Takes a camword as input and returns any spectrograph represented within that camword. By default this includes partial spectrographs (with one or two cameras represented). But if full_spectros_only is set to True, only spectrographs with all cameras represented are given.

Parameters

- **str. The camword of all cameras.** (*camword,*) –
- **bool. Default is False. Flag to specify if you want all spectrographs with any cameras existing** (*full_spectros_only,*) – in the camword (the default) or if you only want fully populated spectrographs.

Returns spectros, list. A list of integer spectrograph numbers represented in the camword input.

desispec.io.util.**camword_union**(*camwords*, *full_spectros_only=False*)
Returns the union of a list of camwords. Optionally can return only those spectros with complete b, r, and z cameras. Note this intentionally does the union before truncating spectrographs, so two partial camwords can lead to an entire spectrograph,

e.g. [a0b1z1, a3r1z2] -> [a013z2] if full_spectros_only=False [a0b1z1, a3r1z2] -> [a013] if full_spectros_only=True

even through no camword has a complete set of camera 1, a complete set is represented in the union.

Parameters

- **list or array of strings. List of camwords.** (*camwords,*) –
- **bool. True if only complete spectrographs with** (*full_spectros_only,*) – b, r, and z cameras in the funal union should be returned.

Returns

final_camword, str. The final union of all input camwords, where truncation of incomplete spectrographs may or may not be performed based on full_spectros_only.

desispec.io.util.**checkgzip**(*filename*)
Check for existence of filename, with or without .gz extension

Parameters **filename** (*str*) – filename to check for

Returns path of existing file without or without .gz, or raises FileNotFoundError if neither exists

desispec.io.util.**create_camword**(*cameras*)
Function that takes in a list of cameras and creates a succinct listing of all spectrographs in the list with cameras. It uses "a" followed by numbers to mean that "all" (b,r,z) cameras are accounted for for those numbers. b, r, and z represent the camera of the same name. All trailing numbers represent the spectrographs for which that camera exists in the list.

Parameters **cameras** (*1-d array or list*) – iterable containing strings of cameras, e.g. 'b0','r1',...

Returns (str): A string representing all information about the spectrographs/cameras given in the input iterable, e.g. a01234678b59z9

desispec.io.util.**decode_camword**(*camword*)
Function that takes in a succinct listing of all spectrographs and outputs a 1-d numpy array with a list of all spectrograph/camera pairs. It uses "a" followed by numbers to mean that "all" (b,r,z) cameras are accounted for for those numbers. b, r, and z represent the camera of the same name. All trailing numbers represent the spectrographs for which that camera exists in the list.

Parameters **camword** (*str*) – A string representing all information about the spectrographs/cameras e.g. a01234678b59z9

Returns (np.ndarray, 1d): an array containing strings of cameras, e.g. 'b0','r1',...

desispec.io.util.**difference_camwords**(*fullcamword*, *badcamword*)
Returns the difference of two camwords. The second argument cameras are removed from the first argument and the remainer is returned. Smart enough to ignore bad cameras if they don't exist in full camword list.

Parameters

- **str. The camword of all cameras**(*fullcamword,*) –
- **str. A camword defining the bad cameras you don't want to include in the final camword that is output**(*badcamword,*) –

Returns str. A camword of cameras in fullcamword that are not in badcamword.

desispec.io.util.**fitsheader**(*header*)
Convert header into astropy.io.fits.Header object.

header can be:

- None: return a blank Header
- list of (key, value) or (key, (value,comment)) entries
- dict d[key] -> value or (value, comment)
- Header: just return it unchanged

Returns fits.Header object

desispec.io.util.**get_speclog**(*nights*, *rawdir=None*)
Scans raw data headers to return speclog of observations. Slow.

Parameters **nights** – list of YEARMMDD nights to scan

Options: rawdir (str): overrides $DESI_SPECTRO_DATA

Returns Table with columns NIGHT,EXPID,MJD,FLAVOR,OBSTYPE,EXPTIME

Scans headers of rawdir/NIGHT/EXPID/desi-EXPID.fits.fz

desispec.io.util.**get_tempfilename**(*filename*)
Returns unique tempfile in same directory as filename with same extension

Parameters **filename** (*str*) – input filename

Returns unique filename in same directory

Example intended usage:

```
tmpfile = get_tempfile(filename)
table.write(tmpfile)
os.rename(tmpfile, filename)
```

By keeping the same extension as the input file, this preserves the ability of table.write to derive the format to use, and if something goes wrong with the I/O it doesn't leave a corrupted partially written file with the final name. The tempfile includes the PID to provide some race condition protection (the last one do do os.rename wins, but at least different processes won't corrupt each other's files).

desispec.io.util.**header2wave**(*header*)
Converts header keywords into a wavelength grid.

returns wave = CRVAL1 + range(NAXIS1)*CDELT1

if LOGLAM keyword is present and true/non-zero, returns 10**wave

desispec.io.util.**healpix_subdirectory**(*nside*, *pixel*)
Return a fixed directory path for healpix grouped files.

Given an NSIDE and NESTED pixel index, return a directory named after a degraded NSIDE and pixel followed by the original nside and pixel. This function is just to ensure that this subdirectory path is always created by the same code.

Parameters

- **nside** (*int*) – a valid NSIDE value.
- **pixel** (*int*) – the NESTED pixel index.

Returns (str): a path containing the low and high resolution directories.

desispec.io.util.**is_svn_current**(*dirname*)
Return True/False for if svn checkout dirname is up-to-date with server

Raises ValueError if unable to determine (e.g. dirname isn't svn checkout)

desispec.io.util.**iterfiles**(*root*, *prefix*, *suffix=None*)
Returns iterator over files starting with *prefix* found under *root* dir Optionally also check if filename ends with *suffix*

desispec.io.util.**makepath**(*outfile*, *filetype=None*)
Create path to outfile if needed.

If outfile isn't a string and filetype is set, interpret outfile as a tuple of parameters to locate the file via findfile().

Returns /path/to/outfile

TODO: maybe a different name?

desispec.io.util.**native_endian**(*data*)
Convert numpy array data to native endianness if needed.

Returns new array if endianness is swapped, otherwise returns input data

Context: By default, FITS data from astropy.io.fits.getdata() are not Intel native endianness and scipy 0.14 sparse matrices have a bug with non-native endian data.

desispec.io.util.**parse_badamps**(*badamps*, *joinsymb=', '*)
Parses badamps string from an exposure or processing table into the (camera,petal,amplifier) sets, with appropriate checking of those values to make sure they're valid. Returns empty list if badamps is None.

Parameters

- **str. A string of {camera}{petal}{amp} entries separated by symbol given with joinsymb (comma** (*badamps,*) – by default). I.e. [brz][0-9][ABCD]. Example: 'b7D,z8A'.

- **str. The symbol separating entries in the str list given by badamps.** (*joinsymb,*) –

Returns

cam_petal_amps, list. A list where each entry is a length 3 tuple of (camera,petal,amplifier). Camera is a lowercase string in [b, r, z]. Petal is an int from 0 to 9. Amplifier is an upper case string in [A, B, C, D].

desispec.io.util.**parse_cameras**(*cameras*, *loglevel='INFO'*)

Function that takes in a representation of all spectrographs and outputs a string that succinctly lists all spectrograph/camera pairs. It uses "a" followed by numbers to mean that "all" (b,r,z) cameras are accounted for for those numbers. b, r, and z represent the camera of the same name. All trailing numbers represent the spectrographs for which that camera exists in the list.

Parameters **str. 1-d array, list** (*cameras,*) – Either a str that is a comma separated list or a series of spectrographs. Also accepts a list or iterable that is processed with create_camword().

Options: loglevel, str: use e.g. "WARNING" to avoid INFO-level log messages for just this call

Returns (str):

camword, str. A string representing all information about the spectrographs/cameras given in the input iterable, e.g. a01234678b59z9

desispec.io.util.**replace_prefix**(*filepath*, *oldprefix*, *newprefix*)

Replace filename prefix even if prefix is elsewhere in path or filename

Parameters

- **filepath** – filename, optionally including path
- **oldprefix** – original prefix to filename part of filepath
- **newprefix** – new prefix to use for filename

Returns new filepath with replaced prefix

e.g. replace_prefix('/blat/foo/blat-bar-blat.fits', 'blat', 'quat') returns '/blat/foo/quat-bar-blat.fits'

desispec.io.util.**validate_badamps**(*badamps*, *joinsymb=','*)

Checks (and transforms) badamps string for consistency with the for need in an exposure or processing table for use in the Pipeline. Specifically ensure they come in (camera,petal,amplifier) sets, with appropriate checking of those values to make sure they're valid. Returns the input string except removing whitespace and replacing potential character separaters with joinsymb (default ','). Returns None if None is given.

Parameters

- **str. A string of {camera}{petal}{amp} entries separated by symbol given with joinsymb (comma** (*badamps,*) – by default). I.e. [brz][0-9][ABCD]. Example: 'b7D,z8A'.
- **str. The symbol separating entries in the str list given by badamps.** (*joinsymb,*) –

Returns

newbadamps, str. Input badamps string of {camera}{petal}{amp} entries separated by symbol given with joinsymb (comma by default). I.e. [brz][0-9][ABCD]. Example: 'b7D,z8A'. Differs from input in that other symbols used to separate terms are replaced by joinsymb and whitespace is removed.

desispec.io.util.**write_bintable**(*filename*, *data*, *header=None*, *comments=None*, *units=None*, *extname=None*, *clobber=False*, *primary_extname='PRIMARY'*)
Utility function to write a fits binary table complete with comments and units in the FITS header too. DATA can either be dictionary, an Astropy Table, a numpy.recarray or a numpy.ndarray.

1.8.26 desispec.io.xytraceset

I/O routines for XYTraceSet objects

desispec.io.xytraceset.**read_xytraceset**(*filename*)
Reads traces in PSF fits file

Parameters **filename** – Path to input fits file which has to contain XTRACE and YTRACE HDUs

Returns XYTraceSet object

desispec.io.xytraceset.**write_xytraceset**(*outfile*, *xytraceset*)
Write a traceset fits file and returns path to file written.

Parameters

- **outfile** – full path to output file
- **xytraceset** – desispec.xytraceset.XYTraceSet object

Returns full filepath of output file that was written

1.8.27 desispec.linalg

Some linear algebra functions.

desispec.linalg.**cholesky_invert**(*A*)
returns the inverse of a positive definite matrix

Args : A : 2D (real symmetric) (nxn) positive definite matrix (numpy.ndarray)

Returns 2D positive definite matrix, inverse of A (numpy.ndarray)

Return type cov

desispec.linalg.**cholesky_solve**(*A*, *B*, *overwrite=False*, *lower=False*)
Returns the solution X of the linear system A.X=B assuming A is a positive definite matrix

Args : A : 2D (real symmetric) (nxn) positive definite matrix (numpy.ndarray) B : 1D vector, must have dimension n (numpy.ndarray)

Options : overwrite: replace A data by cholesky decomposition (faster) lower: cholesky decomposition triangular matrix is lower instead of upper

Returns : X : 1D vector, same dimension as B (numpy.ndarray)

desispec.linalg.**cholesky_solve_and_invert**(*A*, *B*, *overwrite=False*, *lower=False*)
returns the solution X of the linear system A.X=B assuming A is a positive definite matrix

Args : A : 2D (real symmetric) (nxn) positive definite matrix (numpy.ndarray) B : 1D vector, must have dimension n (numpy.ndarray)

Options : overwrite: replace A data by cholesky decomposition (faster) lower: cholesky decomposition triangular matrix is lower instead of upper

Returns X,cov, where X : 1D vector, same dimension n as B (numpy.ndarray) cov : 2D positive definite matrix, inverse of A (numpy.ndarray)

desispec.linalg.**spline_fit**(*output_wave*, *input_wave*, *input_flux*, *required_resolution*, *input_ivar=None*, *order=3*, *max_resolution=None*)

Performs spline fit of input_flux vs. input_wave and resamples at output_wave

Parameters

- **output_wave** – 1D array of output wavelength samples
- **input_wave** – 1D array of input wavelengths
- **input_flux** – 1D array of input flux density
- **required_resolution** (*float*) – resolution for spline knot placement (same unit as wavelength)

Options: input_ivar : 1D array of weights for input_flux order (int) : spline order max_resolution (float) : if not None and first fit fails, try once this resolution

Returns 1D array of flux sampled at output_wave

Return type output_flux

1.8.28 desispec.log

This is a transitional dummy wrapper on desiutil.log.

desispec.log.**get_logger**(**args*, ***kwargs*)

Transitional dummy wrapper on desiutil.log.get_logger().

1.8.29 desispec.maskbits

Mask bits for the spectro pipeline.

Stephen Bailey, LBNL, January 2015

Example:

```
from desispec.maskbits import ccdmask

#- bit operations
mask |= ccdmask.COSMIC      #- set ccdmask.COSMIC in integer/array `mask`
mask & ccdmask.COSMIC       #- get ccdmask.COSMIC from integer/array `mask`
(mask & ccdmask.COSMIC) != 0  #- test boolean status of ccdmask.COSMIC in integer/
↪array `mask`
ccdmask.COSMIC | specmask.SATURATED  #- Combine two bitmasks.

#- bit attributes
ccdmask.mask('COSMIC')      #- 2**0, same as ccdmask.COSMIC
ccdmask.mask(0)             #- 2**0, same as ccdmask.COSMIC
ccdmask.COSMIC              #- 2**0, same as ccdmask.mask('COSMIC')
ccdmask.bitnum('COSMIC')    #- 0
ccdmask.bitname(0)          #- 'COSMIC'
ccdmask.names()             #- ['COSMIC', 'HOT', 'DEAD', 'SATURATED', ...]
ccdmask.names(3)            #- ['COSMIC', 'HOT']
```

(continues on next page)

(continued from previous page)

```
ccdmask.comment(0)          #- "Cosmic ray"
ccdmask.comment('BADPIX')   #- "Cosmic ray"
```

1.8.30 desispec.parallel

Helper functions and classes for dealing with parallelization and related topics.

desispec.parallel.**default_nproc = 1**
Default number of multiprocessing processes. Set globally on first import.

desispec.parallel.**dist_balanced**(*nwork*, *maxworkers*)
Distribute items between a flexible number of workers.

This assumes that each item has equal weight, and that they should be divided into contiguous blocks of items and assigned to workers in rank order.

If the number of workers is less than roughly sqrt(nwork), then we do not reduce the number of workers and the result is the same as the dist_uniform function. If there are more workers than this, then the number of workers is reduced until all workers have close to the same number of tasks.

Parameters

- **nwork** (*int*) – The number of work items.
- **maxworkers** (*int*) – The maximum number of workers. The actual number may be less than this.

Returns A list of tuples, one for each worker. The first element of the tuple is the first item assigned to the worker, and the second element is the number of items assigned to the worker.

desispec.parallel.**dist_discrete**(*worksizes*, *nworkers*, *workerid*, *power=1.0*)
Distribute indivisible blocks of items between groups.

Given some contiguous blocks of items which cannot be subdivided, distribute these blocks to the specified number of groups in a way which minimizes the maximum total items given to any group. Optionally weight the blocks by a power of their size when computing the distribution.

This is effectively the "Painter"s Partition Problem".

Parameters

- **worksizes** (*list*) – The sizes of the indivisible blocks.
- **nworkers** (*int*) – The number of workers.
- **workerid** (*int*) – The worker ID whose range should be returned.
- **power** (*float*) – The power to use for weighting

Returns A tuple. The first element of the tuple is the first block assigned to the worker ID, and the second element is the number of blocks assigned to the worker.

desispec.parallel.**dist_discrete_all**(*worksizes*, *nworkers*, *power=1.0*)
Distribute indivisible blocks of items between groups.

Given some contiguous blocks of items which cannot be subdivided, distribute these blocks to the specified number of groups in a way which minimizes the maximum total items given to any group. Optionally weight the blocks by a power of their size when computing the distribution.

This is effectively the "Painter"s Partition Problem".

Parameters

- **worksizes** (*list*) – The sizes of the indivisible blocks.
- **nworkers** (*int*) – The number of workers.
- **pow** (*float*) – The power to use for weighting

Returns

list of length nworkers; each element is a list of indices of worksizes assigned to that worker.

desispec.parallel.**dist_uniform**(*nwork*, *nworkers*, *id=None*)

Statically distribute some number of items among workers.

This assumes that each item has equal weight, and that they should be divided into contiguous blocks of items and assigned to workers in rank order.

This function returns the index of the first item and the number of items for the specified worker ID, or the information for all workers.

Parameters

- **nwork** (*int*) – the number of things to distribute.
- **nworkers** (*int*) – the number of workers.
- **id** (*int*) – optionally return just the tuple associated with this worker

Returns (tuple): A tuple of ints, containing the first item and number of items. If id=None, then return a list containing the tuple for all workers.

desispec.parallel.**stdouterr_redirected**(*to=None*, *comm=None*)

Redirect stdout and stderr to a file.

The general technique is based on:

http://stackoverflow.com/questions/5081657 http://eli.thegreenplace.net/2015/redirecting-all-kinds-of-stdout-in-python/

One difference here is that each process in the communicator redirects to a different temporary file, and the upon exit from the context the rank zero process concatenates these in order to the file result.

Parameters

- **to** (*str*) – The output file name.
- **comm** (*mpi4py.MPI.Comm*) – The optional MPI communicator.

desispec.parallel.**take_turns**(*comm*, *at_a_time*, *func*, **args*, ***kwargs*)

Processes call a function in groups.

Any extra positional and keyword arguments are passed to the function.

Parameters

- **comm** – mpi4py.MPI.Comm or None.
- **at_a_time** (*int*) – the maximum number of processes to run at a time.
- **func** – the function to call.

Returns The return value on each process is the return value of the function.

desispec.parallel.**use_mpi**()

Return whether we can use MPI.

desispec.parallel.**weighted_partition**(*weights*, *n*, *groups_per_node=None*)
Partition *weights* into *n* groups with approximately same sum(weights)

Parameters

- **weights** – array-like weights
- **n** – number of groups

Returns list of lists of indices of weights for each group

Notes

compared to *dist_discrete_all*, this function allows non-contiguous items to be grouped together which allows better balancing.

1.8.31 desispec.pipeline

Tools for pipeline creation and running.

1.8.32 desispec.pipeline.control

Tools for controling pipeline production.

desispec.pipeline.control.**chain**(*tasktypes*, *nightstr=None*, *states=None*, *expid=None*, *spec=None*, *pack=False*, *nosubmitted=False*, *depjobs=None*, *nersc=None*, *nersc_queue='regular'*, *nersc_maxtime=0*, *nersc_maxnodes=0*, *nersc_shifter=None*, *mpi_procs=1*, *mpi_run=''*, *procs_per_node=0*, *nodb=False*, *out=None*, *debug=False*, *dryrun=False*)
Run a chain of jobs for multiple pipeline steps.

For the list of task types, get all ready tasks meeting the selection criteria. Then either pack all tasks into one job or submit each task type as its own job. Input job dependencies can be specified, and dependencies are tracked between jobs in the chain.

Parameters

- **tasktypes** (*list*) – list of valid task types.
- **nightstr** (*str*) – Comma separated (YYYYMMDD) or regex pattern. Only nights matching these patterns will be considered.
- **states** (*list*) – list of task states to select.
- **nights** (*list*) – list of nights to select.
- **expid** (*int*) – exposure ID to select.
- **pack** (*bool*) – if True, pack all tasks into a single job.
- **nosubmitted** (*bool*) – if True, do not run jobs that have already been submitted.
- **depjobs** (*list*) – list of job ID dependencies.
- **nersc** (*str*) – if not None, the name of the nersc machine to use (cori-haswell | cori-knl).
- **nersc_queue** (*str*) – the name of the queue to use (regular | debug | realtime).
- **nersc_maxtime** (*int*) – if specified, restrict the runtime to this number of minutes.

- **nersc_maxnodes** (*int*) – if specified, restrict the job to use this number of nodes.
- **nersc_shifter** (*str*) – the name of the shifter image to use.
- **mpi_run** (*str*) – if specified, and if not using NERSC, use this command to launch MPI executables in the shell scripts. Default is to not use MPI.
- **mpi_procs** (*int*) – if not using NERSC, the number of MPI processes to use in shell scripts.
- **procs_per_node** (*int*) – if specified, use only this number of processes per node. Default runs one process per core.
- **nodb** (*bool*) – if True, do not use the production DB.
- **out** (*str*) – Put task scripts and logs in this directory relative to the production 'scripts' directory. Default puts task directory in the main scripts directory.
- **debug** (*bool*) – if True, enable DEBUG log level in generated scripts.
- **dryrun** (*bool*) – if True, do not submit the jobs.

Returns the job IDs from the final step in the chain.

Return type list

desispec.pipeline.control.**check_tasks**(*tasks*, *db=None*)

Check the state of pipeline tasks.

If the database handle is given, use the DB for checking. Otherwise use the filesystem.

Parameters

- **tasks** (*list*) – list of tasks to check.
- **db** (DataBase) – the database to use.

Returns dictionary of the state of each task.

Return type OrderedDict

desispec.pipeline.control.**cleanup**(*db*, *tasktypes*, *failed=False*, *submitted=False*, *expid=None*)

Clean up stale tasks in the DB.

Parameters

- **db** (DataBase) – the production DB.
- **tasktypes** (*list*) – list of valid task types.
- **failed** (*bool*) – also clear failed states.
- **submitted** (*bool*) – also clear submitted flag.
- **expid** (*int*) – only clean this exposure ID.

desispec.pipeline.control.**create**(*root=None*, *data=None*, *redux=None*, *prod=None*, *force=False*, *basis=None*, *calib=None*, *db_sqlite=False*, *db_sqlite_path=None*, *db_postgres=False*, *db_postgres_host='nerscdb03.nersc.gov'*, *db_postgres_port=5432*, *db_postgres_name='desidev'*, *db_postgres_user='desidev_admin'*, *db_postgres_authorized='desidev_ro'*, *nside=64*)

Create (or re-create) a production.

Parameters

- **root** (*str*) – value to use for DESI_ROOT.

- **data** (*str*) – value to use for DESI_SPECTRO_DATA.
- **redux** (*str*) – value to use for DESI_SPECTRO_REDUX.
- **prod** (*str*) – value to use for SPECPROD.
- **force** (*bool*) – if True, overwrite existing production DB.
- **basis** (*str*) – value to use for DESI_BASIS_TEMPLATES.
- **calib** (*str*) – value to use for DESI_SPECTRO_CALIB.
- **db_sqlite** (*bool*) – if True, use SQLite for the DB.
- **db_sqlite_path** (*str*) – override path to SQLite DB.
- **db_postgres** (*bool*) – if True, use PostgreSQL for the DB.
- **db_postgres_host** (*str*) – PostgreSQL hostname.
- **db_postgres_port** (*int*) – PostgreSQL connection port number.
- **db_postgres_name** (*str*) – PostgreSQL DB name.
- **db_postgres_user** (*str*) – PostgreSQL user name.
- **db_postgres_authorized** (*str*) – Additional PostgreSQL users to authorize.
- **nside** (*int*) – HEALPix nside value used for spectral grouping.

desispec.pipeline.control.**dryrun**(*tasks*, *nersc=None*, *nersc_queue='regular'*, *nersc_maxtime=0*, *nersc_maxnodes=0*, *nersc_shifter=None*, *mpi_procs=1*, *mpi_run=''*, *procs_per_node=0*, *nodb=False*, *db_postgres_user='desidev_ro'*, *force=False*)

Print equivalent command line jobs.

For the specified tasks, print the equivalent stand-alone commands that would be run on each task. A pipeline job calls the internal desispec.scripts entrypoints directly.

Parameters

- **tasks** (*list*) – list of tasks to run.
- **nersc** (*str*) – if not None, the name of the nersc machine to use (cori-haswell | cori-knl).
- **nersc_queue** (*str*) – the name of the queue to use (regular | debug | realtime).
- **nersc_maxtime** (*int*) – if specified, restrict the runtime to this number of minutes.
- **nersc_maxnodes** (*int*) – if specified, restrict the job to use this number of nodes.
- **nersc_shifter** (*str*) – the name of the shifter image to use.
- **mpi_run** (*str*) – if specified, and if not using NERSC, use this command to launch MPI executables in the shell scripts. Default is to not use MPI.
- **mpi_procs** (*int*) – if not using NERSC, the number of MPI processes to use in shell scripts.
- **procs_per_node** (*int*) – if specified, use only this number of processes per node. Default runs one process per core.
- **nodb** (*bool*) – if True, do not use the production DB.
- **db_postgres_user** (*str*) – If using postgres, connect as this user for read-only access"
- **force** (*bool*) – if True, print commands for all tasks, not just the ones in a ready state.

desispec.pipeline.control.**gen_scripts**(*tasks_by_type*, *nersc=None*, *nersc_queue='regular'*, *nersc_maxtime=0*, *nersc_maxnodes=0*, *nersc_shifter=None*, *mpi_procs=1*, *mpi_run=''*, *procs_per_node=0*, *nodb=False*, *out=None*, *debug=False*, *db_postgres_user='desidev_ro'*)

Generate scripts to run tasks of one or more types.

If multiple task type keys are contained in the dictionary, they will be packed into a single batch job.

Parameters

- **tasks_by_type** (*dict*) – each key is the task type and the value is a list of tasks.
- **nersc** (*str*) – if not None, the name of the nersc machine to use (cori-haswell | cori-knl).
- **nersc_queue** (*str*) – the name of the queue to use (regular | debug | realtime).
- **nersc_maxtime** (*int*) – if specified, restrict the runtime to this number of minutes.
- **nersc_maxnodes** (*int*) – if specified, restrict the job to use this number of nodes.
- **nersc_shifter** (*str*) – the name of the shifter image to use.
- **mpi_run** (*str*) – if specified, and if not using NERSC, use this command to launch MPI executables in the shell scripts. Default is to not use MPI.
- **mpi_procs** (*int*) – if not using NERSC, the number of MPI processes to use in shell scripts.
- **procs_per_node** (*int*) – if specified, use only this number of processes per node. Default runs one process per core.
- **nodb** (*bool*) – if True, do not use the production DB.
- **out** (*str*) – Put task scripts and logs in this directory relative to the production 'scripts' directory. Default puts task directory in the main scripts directory.
- **debug** (*bool*) – if True, enable DEBUG log level in generated scripts.
- **db_postgres_user** (*str*) – If using postgres, connect as this user for read-only access"

Returns the generated script files

Return type list

desispec.pipeline.control.**get_tasks**(*db*, *tasktypes*, *nights*, *states=None*, *expid=None*, *spec=None*, *nosubmitted=False*, *taskfile=None*)

Get tasks of multiple types that match certain criteria.

Parameters

- **db** (*DataBase*) – the production DB.
- **tasktypes** (*list*) – list of valid task types.
- **states** (*list*) – list of task states to select.
- **nights** (*list*) – list of nights to select.
- **expid** (*int*) – exposure ID to select.
- **spec** (*int*) – spectrograph to select.
- **nosubmitted** (*bool*) – if True, ignore tasks that were already submitted.

Returns all tasks of all types.

Return type list

desispec.pipeline.control.**get_tasks_type**(*db*, *tasktype*, *states*, *nights*, *expid=None*, *spec=None*)

Get tasks of one type that match certain criteria.

Parameters

- **db** (DataBase) – the production DB.
- **tasktype** (*str*) – a valid task type.
- **states** (*list*) – list of task states to select.
- **nights** (*list*) – list of nights to select.
- **expid** (*int*) – exposure ID to select.
- **spec** (*int*) – spectrograph to select.

Returns list of tasks meeting the criteria.

Return type (list)

desispec.pipeline.control.**getready**(*db*, *nightstr=None*)

Update forward dependencies in the database.

Update database for one or more nights to ensure that forward dependencies know that they are ready to run.

Parameters

- **db** (DataBase) – the production DB.
- **nightstr** (*list*) – comma separated (YYYYMMDD) or regex pattern.

desispec.pipeline.control.**run**(*taskfile*, *nosubmitted=False*, *depjobs=None*, *nersc=None*, *nersc_queue='regular'*, *nersc_maxtime=0*, *nersc_maxnodes=0*, *nersc_shifter=None*, *mpi_procs=1*, *mpi_run=''*, *procs_per_node=0*, *nodb=False*, *out=None*, *debug=False*)

Create job scripts and run them.

This gets tasks from the taskfile and sorts them by type. Then it generates the scripts. Finally, it runs or submits those scripts to the scheduler.

Parameters

- **taskfile** (*str*) – read tasks from this file (if not specified, read from STDIN).
- **nosubmitted** (*bool*) – if True, do not run jobs that have already been submitted.
- **depjobs** (*list*) – list of job ID dependencies.
- **nersc** (*str*) – if not None, the name of the nersc machine to use (cori-haswell | cori-knl).
- **nersc_queue** (*str*) – the name of the queue to use (regular | debug | realtime).
- **nersc_maxtime** (*int*) – if specified, restrict the runtime to this number of minutes.
- **nersc_maxnodes** (*int*) – if specified, restrict the job to use this number of nodes.
- **nersc_shifter** (*str*) – the name of the shifter image to use.
- **mpi_run** (*str*) – if specified, and if not using NERSC, use this command to launch MPI executables in the shell scripts. Default is to not use MPI.
- **mpi_procs** (*int*) – if not using NERSC, the number of MPI processes to use in shell scripts.
- **procs_per_node** (*int*) – if specified, use only this number of processes per node. Default runs one process per core.

- **nodb** (*bool*) – if True, do not use the production DB.
- **out** (*str*) – Put task scripts and logs in this directory relative to the production 'scripts' directory. Default puts task directory in the main scripts directory.
- **debug** (*bool*) – if True, enable DEBUG log level in generated scripts.

Returns the job IDs returned by the scheduler.

Return type list

desispec.pipeline.control.**run_scripts**(*scripts*, *deps=None*, *slurm=False*)

Run job scripts with optional dependecies.

This either submits the jobs to the scheduler or simply runs them in order with subprocess.

Parameters

- **scripts** (*list*) – list of pathnames of the scripts to run.
- **deps** (*list*) – optional list of job IDs which are dependencies for these scripts.
- **slurm** (*bool*) – if True use slurm to submit the jobs.

Returns the job IDs returned by the scheduler.

Return type list

desispec.pipeline.control.**script**(*taskfile*, *nersc=None*, *nersc_queue='regular'*, *nersc_maxtime=0*, *nersc_maxnodes=0*, *nersc_shifter=None*, *mpi_procs=1*, *mpi_run=''*, *procs_per_node=0*, *nodb=False*, *out=None*, *debug=False*, *db_postgres_user='desidev_ro'*)

Generate pipeline scripts for a taskfile.

This gets tasks from the taskfile and sorts them by type. Then it generates the scripts.

Parameters

- **taskfile** (*str*) – read tasks from this file (if not specified, read from STDIN).
- **nersc** (*str*) – if not None, the name of the nersc machine to use (cori-haswell | cori-knl).
- **nersc_queue** (*str*) – the name of the queue to use (regular | debug | realtime).
- **nersc_maxtime** (*int*) – if specified, restrict the runtime to this number of minutes.
- **nersc_maxnodes** (*int*) – if specified, restrict the job to use this number of nodes.
- **nersc_shifter** (*str*) – the name of the shifter image to use.
- **mpi_run** (*str*) – if specified, and if not using NERSC, use this command to launch MPI executables in the shell scripts. Default is to not use MPI.
- **mpi_procs** (*int*) – if not using NERSC, the number of MPI processes to use in shell scripts.
- **procs_per_node** (*int*) – if specified, use only this number of processes per node. Default runs one process per core.
- **nodb** (*bool*) – if True, do not use the production DB.
- **out** (*str*) – Put task scripts and logs in this directory relative to the production 'scripts' directory. Default puts task directory in the main scripts directory.
- **debug** (*bool*) – if True, enable DEBUG log level in generated scripts.
- **db_postgres_user** (*str*) – If using postgres, connect as this user for read-only access"

Returns the generated script files

Return type list

desispec.pipeline.control.**status**(*task=None*, *tasktypes=None*, *nightstr=None*, *states=None*, *expid=None*, *spec=None*, *db_postgres_user='desidev_ro'*)

Check the status of pipeline tasks.

Args:

Returns None

desispec.pipeline.control.**sync**(*db*, *nightstr=None*, *specdone=False*)

Synchronize DB state based on the filesystem.

This scans the filesystem for all tasks for the specified nights, and updates the states accordingly.

Parameters

- **db** (DataBase) – the production DB.
- **nightstr** (*list*) – comma separated (YYYYMMDD) or regex pattern.
- **specdone** – If true, set spectra to done if files exist.

desispec.pipeline.control.**tasks**(*tasktypes*, *nightstr=None*, *states=None*, *expid=None*, *spec=None*, *nosubmitted=False*, *db_postgres_user='desidev_ro'*, *taskfile=None*)

Get tasks of multiple types that match certain criteria.

Parameters

- **tasktypes** (*list*) – list of valid task types.
- **nightstr** (*list*) – comma separated (YYYYMMDD) or regex pattern.
- **states** (*list*) – list of task states to select.
- **expid** (*int*) – exposure ID to select.
- **spec** (*int*) – spectrograph to select.
- **nosubmitted** (*bool*) – if True, ignore tasks that were already submitted.
- **db_postgres_user** (*str*) – If using postgres, connect as this user for read-only access"
- **taskfile** (*str*) – if set write to this file, else write to STDOUT.

desispec.pipeline.control.**update**(*nightstr=None*, *nside=64*, *expid=None*)

Update a production.

Parameters

- **nightstr** (*str*) – Comma separated (YYYYMMDD) or regex pattern. Only nights matching these patterns will be considered.
- **nside** (*int*) – HEALPix nside value used for spectral grouping.
- **expid** (*int*) – Only update the production for a single exposure ID.

1.8.33 desispec.pipeline.db

Pipeline processing database

class desispec.pipeline.db.**DataBase**

Class for tracking pipeline processing objects and state.

cleanup (*tasktypes=None*, *expid=None*, *cleanfailed=False*, *cleansubmitted=False*)
Reset states of tasks.

Any tasks that are marked as "running" will have their state reset to "ready". This can be called if a job dies before completing all tasks.

Parameters

- **tasktypes** (*list*) – if not None, clean up only tasks of these types.
- **expid** (*int*) – if not None, only clean tasks related to this exposure ID. Note that tasks which are independent of an expid (psfnight, fiberflatnight, spectra, redshift) will be ignored if this option is given.
- **cleanfailed** (*bool*) – if True, also reset failed tasks to ready.
- **cleansubmitted** (*bool*) – if True, set submitted flag to False.

count_task_states (*tasktype*)
Return a dictionary of how many tasks are in each state

Parameters **tasktype** (*str*) – the type of these tasks.

Returns keyed by state, values are number of tasks in that state0

Return type dict

get_states (*tasks*)
Efficiently get the state of many tasks at once.

Parameters **tasks** (*list*) – list of task names.

Returns the state of each task.

Return type dict

get_states_type (*tasktype*, *tasks*)
Efficiently get the state of many tasks of a single type.

Parameters

- **tasktype** (*str*) – the type of these tasks.
- **tasks** (*list*) – list of task names.

Returns the state of each task.

Return type dict

get_submitted (*tasks*)
Return the submitted flag for the list of tasks.

Parameters **tasks** (*list*) – list of task names.

Returns

the boolean submitted state of each task (True means that the task has been submitted).

Return type (dict)

getready (*night=None*)
Update DB, changing waiting to ready depending on status of dependencies .

Parameters **night** (*str*) – The night to process.

set_states (*tasks*)
Efficiently set the state of many tasks at once.

Parameters **tasks** (*list*) – list of tuples containing the task name and the state to set.

Returns Nothing.

set_states_type(*tasktype*, *tasks*, *postprocessing=True*)
Efficiently get the state of many tasks of a single type.

Parameters

- **tasktype** (*str*) – the type of these tasks.
- **tasks** (*list*) – list of tuples containing the task name and the state to set.

Returns Nothing.

set_submitted(*tasks*, *unset=False*)
Flag a list of tasks as submitted.

Parameters

- **tasks** (*list*) – list of task names.
- **unset** (*bool*) – if True, invert the behavior and unset the submitted flag for these tasks.

Returns Nothing.

set_submitted_type(*tasktype*, *tasks*, *unset=False*)
Flag a list of tasks of a single type as submitted.

Parameters

- **tasktype** (*str*) – the type of these tasks.
- **tasks** (*list*) – list of task names.
- **unset** (*bool*) – if True, invert the behavior and unset the submitted flag for these tasks.

Returns Nothing.

sync(*night*, *specdone=False*)
Update states of tasks based on filesystem.

Go through all tasks in the DB for the given night and determine their state on the filesystem. Then update the DB state to match.

Parameters

- **night** (*str*) – The night to scan for updates.
- **specdone** – If true, set spectra to done if files exist.

update(*night*, *nside*, *expid=None*)
Update DB based on raw data.

This will use the usual io.meta functions to find raw exposures. For each exposure, the fibermap and all following objects will be added to the DB.

Parameters

- **night** (*str*) – The night to scan for updates.
- **nside** (*int*) – The current NSIDE value used for pixel grouping.
- **expid** (*int*) – Only update the DB for this exposure.

class desispec.pipeline.db.**DataBasePostgres**(*host*, *port*, *dbname*, *user*, *schema=None*, *authorize=None*)
Pipeline database using PostgreSQL as the backend.

Parameters

- **host** (*str*) – The database server.
- **port** (*int*) – The connection port.
- **dbname** (*str*) – The database to connect.
- **user** (*str*) – The user name for the connection. The password should be stored in the ~/.pgpass file.
- **schema** (*str*) – The schema within the database. If this is specified, then the database is assumed to exist. Otherwise the schema is computed from a hash of the production location and will be created.
- **authorize** (*str*) – If creating the schema, this is the list of additional roles that should be granted access.

initdb()
Create DB tables for all tasks if they do not exist.

class desispec.pipeline.db.**DataBaseSqlite**(*path*, *mode*)
Pipeline database using sqlite3 as the backend.

Parameters

- **path** (*str*) – the filesystem path of the database to open. If None, then a temporary database is created in memory.
- **mode** (*str*) – if "r", the database is open in read-only mode. If "w", the database is open in read-write mode and created if necessary.

initdb()
Create DB tables for all tasks if they do not exist.

desispec.pipeline.db.**all_task_types**()
Get the list of possible task types that are supported.

Returns The list of supported task types.

Return type list

desispec.pipeline.db.**all_tasks**(*night*, *nside*, *expid=None*)
Get all possible tasks for a single night.

This uses the filesystem to query the raw data for a particular night and return a dictionary containing all possible tasks for each task type. For objects which span multiple nights (e.g. spectra, redrock), this returns the tasks which are touched by the given night.

Parameters

- **night** (*str*) – The night to scan for tasks.
- **nside** (*int*) – The HEALPix NSIDE value to use.
- **expid** (*int*) – Only get tasks for this single exposure.

Returns

a dictionary whose keys are the task types and where each value is a list of task properties.

Return type dict

desispec.pipeline.db.**check_tasks**(*tasklist*, *db=None*, *inputs=None*)
Check a list of tasks and return their state.

If the database is specified, it is used to check the state of the tasks and their dependencies. Otherwise the filesystem is checked.

Parameters

- **tasklist** (*list*) – list of tasks.
- **db** (*pipeline.db.DB*) – The optional database to use.
- **inputs** (*dict*) – optional dictionary containing the only input dependencies that should be considered.

Returns The current state of all tasks.

Return type dict

desispec.pipeline.db.**load_db**(*dbstring*, *mode='w'*, *user=None*)

Load a database from a connection string.

This instantiates either an sqlite or postgresql database using a string. If this string begins with "postgresql:", then it is taken to be the information needed to connect to a postgres server. Otherwise it is assumed to be a filesystem path to use with sqlite. The mode is only meaningful when using sqlite. Postgres permissions are controlled through the user permissions.

Parameters

- **dbstring** (*str*) – either a filesystem path (sqlite) or a colon-separated string of connection properties in the form "postresql:<host>:<port>:<dbname>:<user>:<schema>".
- **mode** (*str*) – for sqlite, the mode.
- **user** (*str*) – for postgresql, an alternate user name for opening the DB. This can be used to connect as a user with read-only access.

Returns a derived database class of the appropriate type.

Return type *DataBase*

desispec.pipeline.db.**task_sort**(*tasks*)

Sort a list of tasks by type.

This takes a list of arbitrary tasks and sorts them by type. The result is placed in an ordered dictionary of lists in run order.

Parameters **tasks** (*list*) – the list of input tasks.

Returns ordered dictionary of tasks sorted by type.

Return type (OrderedDict)

1.8.34 desispec.pipeline.defs

Common definitions needed by pipeline modules.

desispec.pipeline.defs.**prod_options_name = 'options.yaml'**

The name of the options file inside the run directory.

desispec.pipeline.defs.**state_colors = {'done': '#00ff00', 'failed': '#ff0000', 'ready':**

State colors used for visualization.

desispec.pipeline.defs.**task_name_sep = '_'**

The separator string used for building object names.

desispec.pipeline.defs.**task_states = ['waiting', 'ready', 'running', 'done', 'failed']**

The valid states of each pipeline task.

1.8.35 desispec.pipeline.prod

Functions for updating and loading a production.

desispec.pipeline.prod.**load_prod**(*mode='w'*, *user=None*)
Load the database and options for a production.

This loads the database from the production location defined by the usual DESI environment variables. It also loads the global options file for the production.

Parameters

- **mode** (*str*) – open mode for sqlite database ("r" or "w").
- **user** (*str*) – for postgresql, an alternate user name for opening the DB. This can be used to connect as a user with read-only access.

Returns

(pipeline.db.DataBase, dict) The database for the production and the global options dictionary.

Return type tuple

desispec.pipeline.prod.**select_nights**(*allnights*, *nightstr*)
Select nights based on regex matches.

Given a list of nights, select all nights matching the specified patterns and return this subset.

Parameters

- **allnights** (*list*) – list of all nights as strings
- **nightstr** (*str*) – comma-separated list of regex patterns.

Returns list of nights that match the patterns.

Return type list

desispec.pipeline.prod.**task_read**(*path*)
Read a task list from a text file or STDIN.

Lines that begin with '#' are ignored as comments. If the path is None, lines are read from STDIN until an EOF marker is received.

Parameters **path** (*str*) – the input file name.

Returns the list of tasks.

Return type list

desispec.pipeline.prod.**task_write**(*path*, *tasklist*)
Write a task list to a text file or STDOUT.

If the path is None, write lines to STDOUT. In all cases, write a special termination line so that this stream or file can be passed into the task_read function.

Parameters

- **path** (*str*) – the output file name.
- **tasklist** (*list*) – the data.

Returns nothing.

desispec.pipeline.prod.**update_prod**(*nightstr=None*, *hpxnside=64*, *expid=None*)
Create or update a production directory tree.

For a given production, create the directory hierarchy and the starting default options.yaml file if it does not exist. Also initialize the production DB if it does not exist. Then update the DB with one or more nights from the raw data. Nights to update may be specified by a list of simple regex matches.

Parameters

- **nightstr** (*str*) – comma-separated list of regex patterns.
- **hpxnside** (*int*) – The nside value to use for spectral grouping.
- **expid** (*int*) – Only update a single exposure. If this is specified, then nightstr must contain only a single night.

desispec.pipeline.prod.**yaml_read**(*path*)
Read a dictionary from a file.

Parameters **path** (*str*) – the input file name.

Returns the data.

Return type dict

desispec.pipeline.prod.**yaml_write**(*path*, *input*)
Write a dictionary to a file.

Parameters

- **path** (*str*) – the output file name.
- **input** (*dict*) – the data.

Returns nothing.

1.8.36 desispec.pipeline.run

Tools for running the pipeline.

exception desispec.pipeline.run.**TimeoutError**

desispec.pipeline.run.**dry_run**(*tasktype*, *tasklist*, *opts*, *procs*, *procs_per_node*, *db=None*, *launch='mpirun -np'*, *force=False*)
Compute the distribution of tasks and equivalent commands.

This function takes similar arguments as run_task_list() except simulates the data distribution and commands that would be run if given the specified number of processes and processes per node.

This can be used to debug issues with the runtime concurrency or the actual options that will be passed to the underying main() entry points for each task.

This function requires that the DESI environment variables are set to point to the current production directory.

Only tasks that are ready to run (based on the filesystem checks or the database) will actually be attempted.

NOTE: Since this function is just informative and for interactive use, we print information directly to STDOUT rather than logging.

Parameters

- **tasktype** (*str*) – the pipeline step to process.
- **tasklist** (*list*) – the list of tasks. All tasks should be of type "tasktype" above.

- **opts** (*dict*) – the global options (for example, as read from the production options.yaml file).
- **procs** (*int*) – the number of processes to simulate.
- **procs_per_node** (*int*) – the number of processes per node to simulate.
- **db** (*pipeline.db.DB*) – The optional database to update.
- **launch** (*str*) – The launching command for a job. This is just a convenience and prepended to each command before the number of processes.
- **force** (*bool*) – If True, ignore database and filesystem state and just run the tasks regardless.

Returns Nothing.

desispec.pipeline.run.**run_dist**(*tasktype*, *tasklist*, *db*, *nproc*, *procs_per_node*, *force=False*)

Compute the runtime distribution of tasks.

For a given number of processes, parse job environment variables and compute the number of workers to use and the remaining tasks to process. Divide the processes into groups, and associate some (or all) of those groups to workers. Assign tasks to these groups of processes. Some groups may have zero tasks if there are more groups than workers needed.

Returns

The (groupsize, groups, tasks, dist) information. Groupsize is the processes per group. Groups is a list of tuples (one per process) giving the group number and rank within the group. The tasks are a sorted list of tasks containing the subset of the inputs that needs to be run. The dist is a list of tuples (one per group) containing the indices of tasks assigned to each group.

Return type tuple

desispec.pipeline.run.**run_task**(*name*, *opts*, *comm=None*, *logfile=None*, *db=None*)

Run a single task.

Based on the name of the task, call the appropriate run function for that task. Log output to the specified file. Run using the specified MPI communicator and optionally update state to the specified database.

Note: This function DOES NOT check the database or filesystem to see if the task has been completed or if its dependencies exist. It assumes that some higher-level code has done that if necessary.

Parameters

- **name** (*str*) – the name of this task.
- **opts** (*dict*) – options to use for this task.
- **comm** (*mpi4py.MPI.Comm*) – optional MPI communicator.
- **logfile** (*str*) – output log file. If None, do not redirect output to a file.
- **db** (*pipeline.db.DB*) – The optional database to update.

Returns the total number of processes that failed.

Return type int

desispec.pipeline.run.**run_task_list**(*tasktype*, *tasklist*, *opts*, *comm=None*, *db=None*, *force=False*)

Run a collection of tasks of the same type.

This function requires that the DESI environment variables are set to point to the current production directory.

This function first takes the communicator and uses the maximum processes per task to split the communicator and form groups of processes of the desired size. It then takes the list of tasks and uses their relative run time estimates to assign tasks to the process groups. Each process group loops over its assigned tasks.

If the database is not specified, no state tracking will be done and the filesystem will be checked as needed to determine the current state.

Only tasks that are ready to run (based on the filesystem checks or the database) will actually be attempted.

Parameters

- **tasktype** (*str*) – the pipeline step to process.
- **tasklist** (*list*) – the list of tasks. All tasks should be of type "tasktype" above.
- **opts** (*dict*) – the global options (for example, as read from the production options.yaml file).
- **comm** (*mpi4py.Comm*) – the full communicator to use for whole set of tasks.
- **db** (*pipeline.db.DB*) – The optional database to update.
- **force** (*bool*) – If True, ignore database and filesystem state and just run the tasks regardless.

Returns

the number of ready tasks, number that are done, and the number that failed.

Return type tuple

desispec.pipeline.run.**run_task_list_db**(*tasktype*, *tasklist*, *comm=None*)

Run a list of tasks using the pipeline DB and options.

This is a wrapper around run_task_list which uses the production database and global options file.

Parameters

- **tasktype** (*str*) – the pipeline step to process.
- **tasklist** (*list*) – the list of tasks. All tasks should be of type "tasktype" above.
- **comm** (*mpi4py.Comm*) – the full communicator to use for whole set of tasks.

Returns the number of ready tasks, and the number that failed.

Return type tuple

desispec.pipeline.run.**run_task_simple**(*name*, *opts*, *comm=None*)

Run a single task with no DB or log redirection.

This a wrapper around run_task() for use without a database and with no log redirection. See documentation for that function.

Parameters

- **name** (*str*) – the name of this task.
- **opts** (*dict*) – options to use for this task.
- **comm** (*mpi4py.MPI.Comm*) – optional MPI communicator.

Returns the total number of processes that failed.

Return type int

1.8.37 desispec.pipeline.scriptgen

Tools for generating shell and slurm scripts.

desispec.pipeline.scriptgen.**batch_nersc**(*tasks_by_type*, *outroot*, *logroot*, *jobname*, *machine*, *queue*, *maxtime*, *maxnodes*, *nodeprocs=None*, *openmp=False*, *multiproc=False*, *db=None*, *shifterimg=None*, *debug=False*)

Generate slurm script(s) to process lists of tasks.

Given sets of task lists and constraints about the machine, generate slurm scripts for use at NERSC.

Parameters

- **tasks_by_type** (*OrderedDict*) – Ordered dictionary of the tasks for each type to be written to a single job script.
- **outroot** (*str*) – root output script name.
- **logroot** (*str*) – root output log name.
- **jobname** (*str*) – the name of the job.
- **machine** (*str*) – the NERSC machine name.
- **queue** (*str*) – the name of the queue
- **maxtime** (*int*) – the maximum run time in minutes.
- **maxnodes** (*int*) – the maximum number of nodes to use.
- **nodeprocs** (*int*) – the number of processes to use per node.
- **openmp** (*bool*) – if True, set OMP_NUM_THREADS to the correct value.
- **multiproc** (*bool*) – if True, use OMP_NUM_THREADS=1 and disable core binding of processes.
- **db** (DataBase) – the pipeline database handle.
- **shifter** (*str*) – the name of the shifter image to use.
- **debug** (*bool*) – if True, set DESI log level to DEBUG in the script.

Returns list of generated slurm files.

Return type (list)

desispec.pipeline.scriptgen.**batch_shell**(*tasks_by_type*, *outroot*, *logroot*, *mpirun=''*, *mpiprocs=1*, *openmp=1*, *db=None*)

Generate bash script(s) to process lists of tasks.

Given sets of task lists, generate a script that processes each in order.

Parameters

- **tasks_by_type** (*OrderedDict*) – Ordered dictionary of the tasks for each type to be written to a single job script.
- **outroot** (*str*) – root output script name.
- **logroot** (*str*) – root output log name.
- **mpirun** (*str*) – optional command to use for launching MPI programs.
- **mpiprocs** (*int*) – if mpirun is specified, use this number of processes.
- **openmp** (*int*) – value to set for OMP_NUM_THREADS.

- **db** (DataBase) – the pipeline database handle.

Returns list of generated script files.

Return type (list)

desispec.pipeline.scriptgen.**dump_job_env**(*fh*, *tfactor*, *startup*, *nworker*, *workersize*)
Write parameters needed at runtime to an open filehandle.

desispec.pipeline.scriptgen.**nersc_job**(*jobname*, *path*, *logroot*, *desisetup*, *commands*, *machine*, *queue*, *nodes*, *cnodes*, *ppns*, *minutes*, *nworker*, *workersize*, *multisrun=False*, *openmp=False*, *multiproc=False*, *shifterimg=None*, *debug=False*)
Create a SLURM script for use at NERSC.

Args:

desispec.pipeline.scriptgen.**parse_job_env**()
Retrieve job parameters from the environment.

1.8.38 desispec.pipeline.tasks

Classes that describe pipeline tasks.

1.8.39 desispec.pipeline.tasks.base

Common operations for pipeline tasks.

class desispec.pipeline.tasks.base.**BaseTask**
Base class for tasks.

This defines the interfaces for the classes representing pipeline tasks. This class should not be instantiated directly.

_create(*db*)
See BaseTask.create.

_insert(*cursor*, *props*)
See BaseTask.insert.

_retrieve(*db*, *name*)
See BaseTask.retrieve.

_run_max_mem_proc(*name*, *db*)
Return zero (i.e. not a limit)

_run_max_mem_task(*name*, *db*)
Return zero (i.e. no memory requirement)

_state_get(*db*, *name*, *cur=None*)
See BaseTask.state_get.

_state_set(*db*, *name*, *state*, *cur=None*)
See BaseTask.state_set.

create(*db*)
Initialize a database for this task type.

This may include creating one or more tables.

Parameters **db** (*pipeline.DB*) – the database instance.

deps (*name*, *db=None*, *inputs=None*)
Get the dependencies for a task.

This gets a list of other tasks which are required.

Parameters

- **name** (*str*) – the task name.
- **db** (*pipeline.DB*) – the optional database instance.
- **inputs** (*dict*) – optional dictionary containing the only input dependencies that should be considered.

Returns

a dictionary of dependencies. The keys are arbitrary and the values can be either scalar task names or lists of tasks.

Return type dict

getready (*db*, *name*, *cur*)
Checks whether dependencies are ready

insert (*cursor*, *props*)
Insert a task into a database.

This uses the name and extra keywords to update one or more task-specific tables.

Parameters

- **cursor** (*DB cursor*) – the database cursor of an open connection.
- **props** (*dict*) – dictionary of properties for the task.

name_join (*props*)
Construct a task name from its properties.

Parameters **props** (*dict*) – dictionary of properties.

Returns the task name.

Return type str

name_split (*name*)
Split a task name into its properties.

Parameters **name** (*str*) – the task name.

Returns dictionary of properties.

Return type dict

paths (*name*)
The filesystem path(s) associated with this task.

Parameters **name** (*str*) – the task name.

Returns the list of output files generated by this task.

Return type list

postprocessing (*db*, *name*, *cur*)
For successful runs, postprocessing on DB

retrieve (*db*, *name*)
Retrieve all task information from the DB.

This may include additional information beyond the contents of the task name (e.g. from other tables).

Parameters

- **db** (*pipeline.DB*) – the database instance.
- **name** (*str*) – the task name.

Returns dictionary of properties for the task.

Return type dict

run(*name*, *opts*, *comm=None*, *db=None*)
Run the task.

Parameters

- **name** (*str*) – the name of this task.
- **opts** (*dict*) – options to use for this task.
- **comm** (*mpi4py.MPI.Comm*) – optional MPI communicator.
- **db** (*pipeline.db.DB*) – The database.

Returns the number of processes that failed.

Return type int

run_and_update(*db*, *name*, *opts*, *comm=None*)
Run the task and update DB state.

The state of the task is marked as "done" if the command completes without raising an exception and if the output files exist.

Parameters

- **db** (*pipeline.db.DB*) – The database.
- **name** (*str*) – the name of this task.
- **opts** (*dict*) – options to use for this task.
- **comm** (*mpi4py.MPI.Comm*) – optional MPI communicator.

Returns the number of processes that failed.

Return type int

run_cli(*name*, *opts*, *procs*, *launch=None*, *log=None*, *db=None*)
Return the equivalent command-line interface.

Parameters

- **name** (*str*) – the name of the task.
- **opts** (*dict*) – dictionary of runtime options.
- **procs** (*int*) – The number of processes to use.
- **launch** (*str*) – optional launching command.
- **log** (*str*) – optional log file for output.
- **db** (*pipeline.db.DB*) – The database.

Returns a command line.

Return type str

run_defaults()
Default options.

This dictionary of default options will be written to the options.yaml file in a production directory. The options will then be loaded from that file at run time.

Changes to this function will only impact newly-created productions, and these options will be overridden by any changes to the options.yaml file.

Returns dictionary of default options.

Return type dict

run_max_mem_proc(*name*, *db=None*)
Maximum memory in GB per process required.

If zero is returned, it indicates that the memory requirement is so small that the code can run fully-packed on any system.

Parameters

- **name** (*str*) – the name of the task.
- **db** (*pipeline.DB*) – the optional database instance.

Returns the required RAM in GB per process.

Return type float

run_max_mem_task(*name*, *db=None*)
Maximum memory in GB per task required.

If zero is returned, it indicates that the memory requirement is so small that the code can run on a single node.

Parameters

- **name** (*str*) – the name of the task.
- **db** (*pipeline.DB*) – the optional database instance.

Returns the required RAM in GB per process.

Return type float

run_max_procs()
Maximum number of processes supported by this task type.

Parameters **procs_per_node** (*int*) – the number of processes running per node.

Returns the maximum number of processes. Zero indicates no limit.

Return type int

run_time(*name*, *procs*, *db=None*)
Estimated runtime for a task at maximum concurrency.

Parameters

- **name** (*str*) – the name of the task.
- **procs** (*int*) – the total number of processes used for this task.
- **db** (*pipeline.DB*) – the optional database instance.

Returns estimated minutes of run time.

Return type int

state_get(*db*, *name*, *cur=None*)
Get the state of a task.

This should not be called repeatedly for many tasks- it is more efficient to get the state of many tasks in a single custom SQL query.

Parameters

- **db** (*pipeline.DB*) – the database instance.
- **name** (*str*) – the task name.

Returns the state.

Return type str

state_set(*db*, *name*, *state*, *cur=None*)
Set the state of a task.

This should not be called repeatedly if you are setting the state of many tasks. It is more efficient to do that in a single SQL command.

Parameters

- **db** (*pipeline.DB*) – the database instance.
- **name** (*str*) – the task name.

desispec.pipeline.tasks.base.**task_type**(*name*)
Given a task name, find the type from the list of available ones.

Parameters **name** (*str*) – the name of the task.

Returns the type of the task.

Return type str

class desispec.pipeline.tasks.cframe.**TaskCFrame**
Class containing the properties of a sky fit task.

_deps(*name*, *db*, *inputs*)
See BaseTask.deps.

_option_list(*name*, *opts*)
Build the full list of options.

This includes appending the filenames and incorporating runtime options.

_paths(*name*)
See BaseTask.paths.

_run(*name*, *opts*, *comm*, *db*)
See BaseTask.run.

_run_cli(*name*, *opts*, *procs*, *db*)
See BaseTask.run_cli.

_run_defaults()
See BaseTask.run_defaults.

_run_max_procs()
See BaseTask.run_max_procs.

_run_time(*name*, *procs*, *db*)
See BaseTask.run_time.

postprocessing(*db*, *name*, *cur*)
For successful runs, postprocessing on DB

class desispec.pipeline.tasks.extract.**TaskExtract**
Class containing the properties of one extraction task.

_deps(*name*, *db*, *inputs*)
See BaseTask.deps.

_option_list(*name*, *opts*)
Build the full list of options.

This includes appending the filenames and incorporating runtime options.

_paths(*name*)
See BaseTask.paths.

_run(*name*, *opts*, *comm*, *db*)
See BaseTask.run.

_run_cli(*name*, *opts*, *procs*, *db*)
See BaseTask.run_cli.

_run_defaults()
See BaseTask.run_defaults.

postprocessing(*db*, *name*, *cur*)
For successful runs, postprocessing on DB

class desispec.pipeline.tasks.fiberflat.**TaskFiberflat**
Class containing the properties of one extraction task.

_deps(*name*, *db*, *inputs*)
See BaseTask.deps.

_option_list(*name*, *opts*)
Build the full list of options.

This includes appending the filenames and incorporating runtime options.

_paths(*name*)
See BaseTask.paths.

_run(*name*, *opts*, *comm*, *db*)
See BaseTask.run.

_run_cli(*name*, *opts*, *procs*, *db*)
See BaseTask.run_cli.

_run_defaults()
See BaseTask.run_defaults.

postprocessing(*db*, *name*, *cur*)
For successful runs, postprocessing on DB

class desispec.pipeline.tasks.fiberflatnight.**TaskFiberflatNight**
Class containing the properties of one fiberflat combined night task.

_deps(*name*, *db*, *inputs*)
See BaseTask.deps.

_option_list(*name*, *opts*)
Build the full list of options.

This includes appending the filenames and incorporating runtime options.

_paths(*name*)
See BaseTask.paths.

_run(*name*, *opts*, *comm*, *db*)
See BaseTask.run.

_run_cli(*name*, *opts*, *procs*, *db*)
See BaseTask.run_cli.

_run_defaults()
See BaseTask.run_defaults.

getready(*db*, *name*, *cur*)
Checks whether dependencies are ready

postprocessing(*db*, *name*, *cur*)
For successful runs, postprocessing on DB

class desispec.pipeline.tasks.fibermap.**TaskFibermap**
Class containing the properties of one fibermap.

Since fibermaps have no dependencies and are not created by the pipeline, this class is just used to specify names, etc.

_deps(*name*, *db*, *inputs*)
See BaseTask.deps.

_paths(*name*)
See BaseTask.paths.

_run(*name*, *opts*, *comm*, *db*)
See BaseTask.run.

_run_cli(*name*, *opts*, *procs*, *db*)
See BaseTask.run_cli.

_run_defaults()
See BaseTask.run_defaults.

class desispec.pipeline.tasks.fluxcalib.**TaskFluxCalib**
Class containing the properties of a sky fit task.

_deps(*name*, *db*, *inputs*)
See BaseTask.deps.

_option_list(*name*, *opts*)
Build the full list of options.

This includes appending the filenames and incorporating runtime options.

_paths(*name*)
See BaseTask.paths.

_run(*name*, *opts*, *comm*, *db*)
See BaseTask.run.

_run_cli(*name*, *opts*, *procs*, *db*)
See BaseTask.run_cli.

_run_defaults()
See BaseTask.run_defaults.

postprocessing(*db*, *name*, *cur*)
For successful runs, postprocessing on DB

class desispec.pipeline.tasks.preproc.**TaskPreproc**
Class containing the properties of one preprocessed pixel file.

_deps(*name*, *db*, *inputs*)
See BaseTask.deps.

_option_list(*name*, *opts*)
Build the full list of options.

This includes appending the filenames and incorporating runtime options.

_paths(*name*)
See BaseTask.paths.

_run(*name*, *opts*, *comm*, *db*)
See BaseTask.run.

_run_cli(*name*, *opts*, *procs*, *db*)
See BaseTask.run_cli.

_run_defaults()
See BaseTask.run_defaults.

postprocessing(*db*, *name*, *cur*)
For successful runs, postprocessing on DB

class desispec.pipeline.tasks.psf.**TaskPSF**
Class containing the properties of one PSF task.

_deps(*name*, *db*, *inputs*)
See BaseTask.deps.

_option_list(*name*, *opts*)
Build the full list of options.

This includes appending the filenames and incorporating runtime options.

_paths(*name*)
See BaseTask.paths.

_run(*name*, *opts*, *comm*, *db*)
See BaseTask.run.

_run_cli(*name*, *opts*, *procs*, *db*)
See BaseTask.run_cli.

_run_defaults()
See BaseTask.run_defaults.

postprocessing(*db*, *name*, *cur*)
For successful runs, postprocessing on DB

class desispec.pipeline.tasks.psfnight.**TaskPSFNight**
Class containing the properties of one PSF combined night task.

_deps(*name*, *db*, *inputs*)
See BaseTask.deps.

_option_dict(*name*, *opts*)
Build the full list of options.

This includes appending the filenames and incorporating runtime options.

_option_list(*name*, *opts*)
Build the full list of options.

This includes appending the filenames and incorporating runtime options.

_paths(*name*)
See BaseTask.paths.

_run(*name*, *opts*, *comm*, *db*)
See BaseTask.run.

_run_cli(*name*, *opts*, *procs*, *db*)
See BaseTask.run_cli.

_run_defaults()
See BaseTask.run_defaults.

getready(*db*, *name*, *cur*)
Checks whether dependencies are ready

postprocessing(*db*, *name*, *cur*)
For successful runs, postprocessing on DB

class desispec.pipeline.tasks.qadata.**TaskQAData**
Class containing the properties of a sky fit task.

_deps(*name*, *db*, *inputs*)
See BaseTask.deps.

_option_list(*name*, *opts*)
Build the full list of options.

This includes appending the filenames and incorporating runtime options.

_paths(*name*)
See BaseTask.paths.

_run(*name*, *opts*, *comm*, *db*)
See BaseTask.run.

_run_cli(*name*, *opts*, *procs*, *db*)
See BaseTask.run_cli.

_run_defaults()
See BaseTask.run_defaults.

postprocessing(*db*, *name*, *cur*)
For successful runs, postprocessing on DB

class desispec.pipeline.tasks.rawdata.**TaskRawdata**
Class containing the properties of one rawdata.

Since rawdatas have no dependencies and are not created by the pipeline, this class is just used to specify names, etc.

_deps(*name*, *db*, *inputs*)
See BaseTask.deps.

_paths(*name*)
See BaseTask.paths.

_run(*name*, *opts*, *comm*, *db*)
See BaseTask.run.

_run_cli(*name*, *opts*, *procs*, *db*)
See BaseTask.run_cli.

_run_defaults()
See BaseTask.run_defaults.

class desispec.pipeline.tasks.redshift.**TaskRedshift**
Class containing the properties of one spectra task.

_deps(*name*, *db*, *inputs*)
See BaseTask.deps.

_option_list(*name*, *opts*)
Build the full list of options.

This includes appending the filenames and incorporating runtime options.

_paths(*name*)
See BaseTask.paths.

_run(*name*, *opts*, *comm*, *db*)
See BaseTask.run.

_run_cli(*name*, *opts*, *procs*, *db*)
See BaseTask.run_cli.

_run_defaults()
See BaseTask.run_defaults.

run_and_update(*db*, *name*, *opts*, *comm=None*)
Run the task and update DB state.

The state of the task is marked as "done" if the command completes without raising an exception and if the output files exist.

It is specific for redshift because the healpix_frame table has to be updated

Parameters

- **db** (*pipeline.db.DB*) – The database.
- **name** (*str*) – the name of this task.
- **opts** (*dict*) – options to use for this task.
- **comm** (*mpi4py.MPI.Comm*) – optional MPI communicator.

Returns the number of processes that failed.

Return type int

class desispec.pipeline.tasks.sky.**TaskSky**
Class containing the properties of a sky fit task.

_deps(*name*, *db*, *inputs*)
See BaseTask.deps.

_option_list(*name*, *opts*)
Build the full list of options.

This includes appending the filenames and incorporating runtime options.

_paths(*name*)
See BaseTask.paths.

_run(*name*, *opts*, *comm*, *db*)
See BaseTask.run.

_run_cli(*name*, *opts*, *procs*, *db*)
See BaseTask.run_cli.

_run_defaults()
See BaseTask.run_defaults.

postprocessing(*db*, *name*, *cur*)
For successful runs, postprocessing on DB

class desispec.pipeline.tasks.spectra.**TaskSpectra**
Class containing the properties of one spectra task.

_deps(*name*, *db*, *inputs*)
See BaseTask.deps.

_paths(*name*)
See BaseTask.paths.

_run(*name*, *opts*, *comm*, *db*)
See BaseTask.run.

_run_cli(*name*, *opts*, *procs*, *db*)
See BaseTask.run_cli.

_run_defaults()
See BaseTask.run_defaults.

run_and_update(*db*, *name*, *opts*, *comm=None*)
Run the task and update DB state.

The state of the task is marked as "done" if the command completes without raising an exception and if the output files exist.

It is specific for spectra because the healpix_frame table has to be updated

Parameters

- **db** (*pipeline.db.DB*) – The database.
- **name** (*str*) – the name of this task.
- **opts** (*dict*) – options to use for this task.
- **comm** (*mpi4py.MPI.Comm*) – optional MPI communicator.

Returns the number of processes that failed.

Return type int

class desispec.pipeline.tasks.starfit.**TaskStarFit**
Class containing the properties of one extraction task.

_deps(*name*, *db*, *inputs*)
See BaseTask.deps.

_option_list(*name*, *opts*)
Build the full list of options.

This includes appending the filenames and incorporating runtime options.

_paths(*name*)
See BaseTask.paths.

_run(*name*, *opts*, *comm*, *db*)
See BaseTask.run.

_run_cli(*name*, *opts*, *procs*, *db*)
See BaseTask.run_cli.

_run_defaults()
See BaseTask.run_defaults.

postprocessing(*db*, *name*, *cur*)
For successful runs, postprocessing on DB

class desispec.pipeline.tasks.traceshift.**TaskTraceShift**
Class containing the properties of one trace shift task.

_deps(*name*, *db*, *inputs*)
See BaseTask.deps.

_option_list(*name*, *opts*)
Build the full list of options.

This includes appending the filenames and incorporating runtime options.

_paths(*name*)
See BaseTask.paths.

_run(*name*, *opts*, *comm*, *db*)
See BaseTask.run.

_run_cli(*name*, *opts*, *procs*, *db*)
See BaseTask.run_cli.

_run_defaults()
See BaseTask.run_defaults.

postprocessing(*db*, *name*, *cur*)
For successful runs, postprocessing on DB

Tools to regroup spectra in individual exposures by healpix on the sky

class desispec.pixgroup.**FrameLite**(*wave*, *flux*, *ivar*, *mask*, *resolution_data*, *fibermap*, *header*, *scores=None*)
Lightweight Frame object for regrouping

This is intended for I/O without the overheads of float32 -> float64 conversion, correcting endianness, etc.

classmethod read(*filename*)
Return FrameLite read from *filename*

class desispec.pixgroup.**SpectraLite**(*bands*, *wave*, *flux*, *ivar*, *mask*, *resolution_data*, *fibermap*, *exp_fibermap=None*, *scores=None*)
Lightweight spectra I/O object for regrouping

num_spectra()
Get the number of spectra contained in this group.

Returns (int): Number of spectra contained in this group.

num_targets()
Get the number of distinct targets.

Returns (int): Number of unique targets with spectra in this object.

classmethod read(*filename*)
Return a SpectraLite object read from *filename*

target_ids()
Return list of unique target IDs.

The target IDs are sorted by the order that they first appear.

Returns (array): an array of integer target IDs.

write(*filename*, *header=None*)
Write this SpectraLite object to *filename*

desispec.pixgroup.**add_missing_frames**(*frames*)
Adds any missing frames with ivar=0 FrameLite objects with correct shape to match those that do exist.

Parameters **frames** – dict of FrameLite objects, keyed by (night,expid,camera)

Modifies *frames* in-place.

Example: if *frames* has keys (2020,1,'b0') and (2020,1,'r0') but not (2020,1,'z0'), this will add a blank FrameLite object for z0.

The purpose of this is to facilitate frames2spectra, which needs *something* for every spectro camera for every exposure that is included.

desispec.pixgroup.**fibermap2tilepix**(*fibermap*, *nside=64*)
Maps fibermap to which healpix are covered by which petals

Parameters **fibermap** – table with columns TARGET_RA, TARGET_DEC, PETAL_LOC

Options: nside (int): nested healpix nside (must be power of 2)

Returns dict petalpix[petal] = list(healpix covered by that petal)

desispec.pixgroup.**frames2spectra**(*frames*, *pix=None*, *nside=64*)
Combine a dict of FrameLite into a SpectraLite for healpix *pix*

Parameters **frames** – dict of FrameLight, keyed by (night, expid, camera)

Options: pix: only include targets in this NESTED healpix pixel number nside: Healpix nside, must be power of 2

Returns SpectraLite object with subset of spectra from frames that are in the requested healpix pixel *pix*

desispec.pixgroup.**get_exp2healpix_map**(*survey=None*, *program=None*, *expfile=None*, *specprod_dir=None*, *strict=False*, *nights=None*, *expids=None*)
Maps exposures to healpixels using preproc/NIGHT/EXPID/tilepix*.json files

Options: survey (str): filter by this survey (main, sv3, sv1, ...) program (str): filter by this FAPRGRM (dark, bright, backup, other) specprod_dir (str): override $DESI_SPECTRO_REDUX/$SPECPROD

TODO...

Returns table with columns NIGHT EXPID SPECTRO HEALPIX

desispec.pixgroup.**update_frame_cache**(*frames*, *framekeys*, *specprod_dir=None*)
Update a cache of FrameLite objects to match requested frameskeys

Parameters

- **frames** – dict of FrameLite objects, keyed by (night, expid, camera)
- **framekeys** – list of desired (night, expid, camera)

Updates *frames* in-place

Notes

frames is dictionary, *framekeys* is list. When finished, the keys of *frames* match the entries in *framekeys*

Preprocess raw DESI exposures

desispec.preproc.**_background**(*image*, *header*, *patch_width=200*, *stitch_width=10*, *stitch=False*)
determine background using a 2D median with square patches of width = width that are interpolated and optionnally try and match the level of amplifiers (does not subtract the background)

Parameters

- **image** (*ndarray*) –
- **image =** (*()* –
- **is used to read CCDSEC** (*header*) –

Options: patch_width (integer) size in pixels of the median square patches stitch_width (integer) width in pixels of amplifier edges to match level of amplifiers stitch : do match level of amplifiers

Returns background image with same shape as input image

desispec.preproc.**_clipped_std_bias**(*nsigma*)
Returns the bias on the standard deviation of a sigma-clipped dataset

Divide by the returned bias to get a corrected value:

```
a = nsigma
bias = sqrt((integrate x^2 exp(-x^2/2), x=-a..a) / (integrate exp(-x^2/2), x=-a..
↪a))
     = sqrt(1 - 2a exp(-a^2/2) / (sqrt(2pi) erf(a/sqrt(2))))
```

See http://www.wolframalpha.com/input/?i=(integrate+x%5E2+exp(-x%5E2%2F2),+x+%3D+-a+to+a)+%2F+(integrate+exp(-x%5E2%2F2),+x%3D-a+to+a)

desispec.preproc.**_global_background**(*image*, *patch_width=200*)
determine background using a 2D median with square patches of width = width that are interpolated (does not subtract the background)

Parameters

- **image** (*ndarray*) –
- **image =** (*()* –

Options: patch_width (integer) size in pixels of the median square patches

Returns background image with same shape as input image

desispec.preproc.**_overscan**(*pix*, *nsigma=5*, *niter=3*)
DEPRECATED: See calc_overscan

desispec.preproc.**_savgol_clipped**(*data*, *window=15*, *polyorder=5*, *niter=0*, *threshold=3.0*)
Simple method to iteratively do a SavGol filter with rejection and replacing rejected pixels by nearest neighbors

Parameters

- **data** (*ndarray*) –

- **window** (*int*) – Window parameter for savgol
- **polyorder** (*int*) –
- **niter** (*int*) –
- **threshold** (*float*) –

Returns:

desispec.preproc.**calc_overscan**(*pix*, *nsigma=5*, *niter=3*)
Calculates overscan, readnoise from overscan image pixels

Parameters **pix** (*ndarray*) – overscan pixels from CCD image

Optional: nsigma (float) : number of standard deviations for sigma clipping niter (int) : number of iterative refits

Returns Mean, sigma-clipped value readnoise (float):

Return type overscan (float)

desispec.preproc.**compute_background_between_fiber_blocks**(*image*, *xyset*)
Computes CCD background between blocks of fibers

Parameters

- **image** – desispec.image.Image object
- **xyset** – desispec.xytraceset.XYTraceSet object

Returns (model, qadict): model: np.array of same shape as image.pix qadict: dictionary of keywords for QA with min/max per amplifier

Notes

Has hardcoded number of blocks and fibers and typical spacing between blocked tuned to DESI spectrographs.

desispec.preproc.**compute_overscan_step**(*overscan_col*, *median_size=7*, *edge_margin=50*)
Compute the overscan step score 'OSTEP' from an array of overscan values averaged per CCD row

Parameters **overscan_col** – 1D numpy.array

Options: median_size (int): window size for median pre-filter of overscan_col edge_margin (int): ignore this number of rows at the CCD edges

Returns OSTEP value (float scalar)

desispec.preproc.**get_amp_ids**(*header*)
Return list of amp names based upon header keywords

desispec.preproc.**get_calibration_image**(*cfinder*, *keyword*, *entry*, *header=None*)
Reads a calibration file

Parameters

- **cfinder** – None or CalibFinder object
- **keyword** – BIAS, MASK, or PIXFLAT

- **entry** – boolean or filename or image if entry==False return False if entry==True use calibration filename from calib. config and read it if entry==str use this for the filename if entry==image return input

Options: header : if not None, update header['CAL...'] = calib provenance

Returns 2D numpy array with calibration image

For the case of keyword='BIAS', check for nightly bias before using default bias in $DESI_SPECTRO_CALIB

desispec.preproc.**numba_mean**(*image_flux*, *image_ivar*, *x*, *hw=3*)
Returns mean of pixels vs. row about x+-hw

Parameters

- **image_flux** – 2D array of CCD image pixels
- **image_ivar** – 2D array of inverse variance of image_flux
- **x** – 1D array of x location per row, len(x) = image_flux.shape[0]

Options: hw (int): halfwidth over which to average

Returns (flux, ivar) 1D arrays with weighted mean and inverse variance of pixels[i, int(x-hw):int(x+hw)+1] per row i

desispec.preproc.**parse_sec_keyword**(*value*)
parse keywords like BIASSECB='[7:56,51:4146]' into python slices

python and FITS have almost opposite conventions,

- FITS 1-indexed vs. python 0-indexed
- FITS upperlimit-inclusive vs. python upperlimit-exclusive
- FITS[x,y] vs. python[y,x]

i.e. BIASSEC2='[7:56,51:4146]' -> (slice(50,4146), slice(6,56))

desispec.preproc.**preproc**(*rawimage*, *header*, *primary_header*, *bias=True*, *dark=True*, *pixflat=True*, *mask=True*, *bkgsub_dark=False*, *nocosmic=False*, *cosmics_nsig=6*, *cosmics_cfudge=3.0*, *cosmics_c2fudge=0.5*, *ccd_calibration_filename=None*, *nocrosstalk=False*, *nogain=False*, *overscan_per_row=False*, *use_overscan_row=False*, *use_savgol=None*, *nodarktrail=False*, *remove_scattered_light=False*, *psf_filename=None*, *bias_img=None*, *model_variance=False*, *no_traceshift=False*, *bkgsub_science=False*)
preprocess image using metadata in header

image = ((rawimage-bias-overscan)*gain)/pixflat

Parameters

- **rawimage** – 2D numpy array directly from raw data file
- **header** – dict-like metadata, e.g. from FITS header, with keywords CAMERA, BIASSECx, DATASECx, CCDSECx where x = A, B, C, D for each of the 4 amplifiers (also supports old naming convention 1, 2, 3, 4).
- **primary_header** – dict-like metadata fit keywords EXPTIME, DOSVER DATE-OBS is also required if bias, pixflat, or mask=True

Optional bias, pixflat, and mask can each be: False: don't apply that step True: use default calibration data for that night ndarray: use that array filename (str or unicode): read HDU 0 and use that

Optional overscan features:

overscan_per_row [bool, Subtract the overscan_col values] row by row from the data.

use_overscan_row [bool, Subtract off the overscan_row] from the data (default: False). Requires ORSEC in the Header

use_savgol [bool, Specify whether to use Savitsky-Golay filter for] the overscan. (default: False). Requires use_overscan_row=True to have any effect.

Optional variance model if model_variance=True Optional background subtraction with median filtering accross the whole CCD if bkgsub_dark=True Optional background subtraction with median filtering between groups of fiber traces if bkgsub_science=True

Optional disabling of cosmic ray rejection if nocosmic=True Optional disabling of dark trail correction if nodarktrail=True

Optional bias image (testing only) may be provided by bias_img=

Optional tuning of cosmic ray rejection parameters: cosmics_nsig: number of sigma above background required cosmics_cfudge: number of sigma inconsistent with PSF required cosmics_c2fudge: fudge factor applied to PSF

Optional fit and subtraction of scattered light

Returns Image object with member variables: pix : 2D preprocessed image in units of electrons per pixel ivar : 2D inverse variance of image mask : 2D mask of image (0=good) readnoise : 2D per-pixel readnoise of image meta : metadata dictionary TODO: define what keywords are included

preprocessing includes the following steps:

- bias image subtraction
- overscan subtraction (from BIASSEC* keyword defined regions)
- readnoise estimation (from BIASSEC* keyword defined regions)
- gain correction (from GAIN* keywords)
- pixel flat correction
- cosmic ray masking
- propagation of input known bad pixel mask
- inverse variance estimation

Notes:

The bias image is subtracted before any other calculation to remove any non-uniformities in the overscan regions prior to calculating overscan levels and readnoise.

The readnoise is an image not just one number per amp, because the pixflat image also affects the interpreted readnoise.

The inverse variance is estimated from the readnoise and the image itself, and thus is biased.

`desispec.preproc.`**`read_bias`**(*filename=None*, *camera=None*, *dateobs=None*)

Return calibration bias filename for camera on dateobs or night

Options: filename : input filename to read camera : e.g. 'b0', 'r1', 'z9' dateobs : DATE-OBS string, e.g. '2018-09-23T08:17:03.988'

Notes

must provide filename, or both camera and dateobs

desispec.preproc.**read_dark**(*filename=None*, *camera=None*, *dateobs=None*, *exptime=None*)
Return dark current 2D image (accounting for exposure time)

Parameters

- **filename** – input filename to read
- **camera** – e.g. 'b0', 'r1', 'z9'
- **dateobs** – DATE-OBS string, e.g. '2018-09-23T08:17:03.988'
- **exptime** – the exposure time of the image in seconds

Notes

must provide filename

desispec.preproc.**read_mask**(*filename=None*, *camera=None*, *dateobs=None*)
Read bad pixel mask image for camera on dateobs.

Options: filename : input filename to read camera : e.g. 'b0', 'r1', 'z9' dateobs : DATE-OBS string, e.g. '2018-09-23T08:17:03.988'

Notes

must provide filename, or both camera and dateobs

desispec.preproc.**read_pixflat**(*filename=None*, *camera=None*, *dateobs=None*)
Read calibration pixflat image for camera on dateobs.

Options: filename : input filename to read camera : e.g. 'b0', 'r1', 'z9' dateobs : DATE-OBS string, e.g. '2018-09-23T08:17:03.988'

Notes

must provide filename, or both camera and dateobs

desispec.preproc.**subtract_peramp_overscan**(*image*, *hdr*)
Subtract per-amp overscan using BIASSEC* keywords

Parameters

- **image** – 2D image array, modified in-place
- **hdr** – FITS header with BIASSEC[ABCD] or BIASSEC[1234] keywords

Note: currently used in desispec.ccdcalib.compute_bias_file to model bias image, but not preproc itself (which subtracts that bias, and has more complex support for row-by-row, col-overscan, etc.)

Module for generating QA HTML

desispec.qa.html.**calib**(*qaprod_dir=None*, *specprod_dir=None*)
Generate HTML to orgainze calib HTML

desispec.qa.html.**calib_exp**(*night*, *expid*, *qaprod_dir=None*)
Geneate HTML for calib exposure PNGs :param night: :param expid:

Returns:

desispec.qa.html.**finish**(*f*, *body*, *links=None*)
Fill in the HTML file and end it :param f: :type f: file :param body: :type body: str :param links: :type links: str, optional

desispec.qa.html.**header**(*title*)

Parameters **title** (*str, optional*) –

desispec.qa.html.**make_exposure**(*night*, *expid*, *qaprod_dir=None*)
Generate HTML for exposure PNGs

Parameters

- **setup** (*str*) –
- **cbset** (*str*) –
- **det** (*int*) –

Returns

- **links** (*str*)
- **body** (*str*)

desispec.qa.html.**make_exposures**(*qaprod_dir=None*)
Generate HTML to organize exposure HTML

Returns

- **links** (*str*)
- **body** (*str*)

desispec.qa.html.**toplevel**(*qaprod_dir=None*)
Generate HTML to top level QA Mainly generates the highest level HTML file which has links to the Exposure and Calib QA.

This also slurps any .png files in the top-level

Parameters

- **setup** (*str*) –
- **cbset** (*str*) –
- **det** (*int*) –

Returns

- **links** (*str*)
- **body** (*str*)

Classes to organize and execute QA for a DESI exposure

Classes to organize and execute QA for a DESI exposure

Classes to organize and execute QA for a DESI exposure

desispec.qa.qa_frame.**qaframe_from_frame**(*frame_file*, *specprod_dir=None*, *make_plots=False*, *qaprod_dir=None*, *output_dir=None*, *clobber=True*)
Generate a qaframe object from an input frame_file name (and night)

Write QA to disk Will also make plots if directed :param frame_file: str :param specprod_dir: str, optional :param qa_dir: str, optional – Location of QA :param make_plots: bool, optional :param output_dir: str, optional

Returns:

Class to organize QA for multiple exposures Likely to only be used as parent of QA_Night or QA_Prod

Class to organize QA for one night of DESI exposures

Module for QA plots

desispec.qa.qa_plots.**brick_redrock**(*outfil*, *zf*, *qabrick*)
QA plots for redrock fits

Parameters

- **outfil** –
- **qabrick** –
- **zf** – ZfindBase object

Returns Stuff?

desispec.qa.qa_plots.**exposure_fiberflat**(*channel*, *expid*, *metric*, *outfile=None*)
Generate an Exposure level plot of a FiberFlat metric :param channel: str, e.g. 'b', 'r', 'z' :param expid: int :param metric: str, allowed entires are: ['meanflux']

Returns:

desispec.qa.qa_plots.**exposure_fluxcalib**(*outfil*, *qa_data*)
QA plots for Flux calibration in an Exposure

Parameters

- **outfil** – str – Name of PDF file
- **qa_data** – dict – QA data, including that of the individual frames

desispec.qa.qa_plots.**exposure_map**(*x*, *y*, *metric*, *mlbl=None*, *outfile=None*, *title=None*, *ax=None*, *fig=None*, *psz=9.0*, *cmap=None*, *vmnx=None*)
Generic method used to generated Exposure level QA One channel at a time

Parameters

- **x** – list or ndarray
- **y** – list or ndarray
- **metric** – list or ndarray
- **mlbl** – str, optional
- **outfile** – str, optional
- **title** – str, optional

desispec.qa.qa_plots.**exposure_s2n**(*qa_exp*, *metric*, *outfile='exposure_s2n.png'*, *verbose=True*, *specprod_dir=None*)
Generate an Exposure level plot of a S/N metric :param qa_exp: QA_Exposure :param metric: str, allowed entires are: ['resid'] :param specprod_dir: str, optional

Returns:

desispec.qa.qa_plots.**frame_fiberflat**(*outfil*, *qaframe*, *frame*, *fiberflat*)
QA plots for fiber flat

Parameters

- **outfil** –
- **qaframe** –
- **frame** –
- **fiberflat** –
- **clobber** – bool, optional

Returns Stuff?

desispec.qa.qa_plots.**frame_fluxcalib**(*outfil*, *qaframe*, *frame*, *fluxcalib*)
QA plots for Flux calibration in a Frame

Parameters

- **outfil** – str, name of output file
- **qaframe** – dict containing QA info
- **frame** – frame object containing extraction of standard stars
- **fluxcalib** – fluxcalib object containing flux calibration

Returns:

desispec.qa.qa_plots.**frame_s2n**(*s2n_dict*, *outfile*, *rescut=0.2*, *verbose=True*)
Plot S/N diagnostics for a given frame Replaces a previous-QL script

Parameters

- **s2n_dict** (*dict*) – dictionary of qa outputs repackaged a bit
- **outfile** (*str*) – output png filename
- **rescut** (*float, optional*) – only plot residuals (+/-) less than rescut

desispec.qa.qa_plots.**frame_skyres**(*outfil*, *frame*, *skymodel*, *qaframe*, *quick_look=False*)
Generate QA plots and files for sky residuals of a given frame

Parameters

- **outfil** (*str*) – Name of output file
- **frame** (*Frame object*) –
- **skymodel** (*SkyModel object*) –
- **qaframe** (*QAFrame object*) –

desispec.qa.qa_plots.**get_channel_clrs**()
Simple dict to organize styles for channels :returns: dict :rtype: channel_dict

desispec.qa.qa_plots.**get_sty_otype**()
Styles for plots

desispec.qa.qa_plots.**prod_ZP**(*qa_prod*, *outfile=None*, *channels=('b', 'r', 'z')*, *xaxis='MJD'*)
Generate a plot summarizing the ZP for a production

Parameters

- **qa_prod** – QA_Prod object
- **outfile** – str, optional Output file name
- **channels** – tuple, optional List of channels to show
- **xaxis** – str, optional Designate x-axis. Options are: MJD, expid, texp

Returns:

desispec.qa.qa_plots.**prod_avg_s2n**(*qa_prod, outfile=None, optypes=['ELG'], xaxis='MJD', fiducials=None*)

Generate a plot summarizing average S/N in a production for a few object types in a few cameras

Parameters

- **qa_prod** – QA_Prod object
- **outfile** – str, optional Output file name
- **optypes** – list, optional List of fiducial objects to show Options are: ELG, LRG, QSO
- **xaxis** – str, optional Designate x-axis. Options are: MJD, expid, texp
- **fiducials** –

Returns:

desispec.qa.qa_plots.**prod_channel_hist**(*qa_prod*, *qatype*, *metric*, *xlim=None*, *outfile=None*, *pp=None*, *close=True*)

Generate a series of histrograms (one per channel)

Parameters

- **qa_prod** – QA_Prod class
- **qatype** – str
- **metric** – str
- **xlim** – tuple, optional
- **outfile** – str, optional
- **pp** – PdfPages, optional
- **close** – bool, optional

Returns:

desispec.qa.qa_plots.**prod_time_series**(*qa_multi*, *qatype*, *metric*, *outfile=None*, *close=True*, *pp=None*, *bright_dark=0*, *exposures=False*, *night=None*, *horiz_line=None*)

Generate a time series plot for a production Can be MJD or Exposure number

Parameters

- **qa_multi** – QA_Prod or QA_Night
- **qatype** – str
- **metric** – str
- **outfile** – str, optional
- **close** – bool, optional
- **pp** –
- **bright_dark** – int, optional; (flag: 0=all; 1=bright; 2=dark)
- **night** – str, optional Only used for the Title
- **horiz_line** – float, optional Draw a horizontal line at input value

desispec.qa.qa_plots.**show_meta**(*ax*, *qaframe*, *qaflavor*, *outfil*)

Show meta data on the figure

Parameters

- **ax** – matplotlib.ax
- **qaframe** – QA_Frame
- **qaflavor** – str

Returns:

desispec.qa.qa_plots.**skyline_resid**(*channel*, *sky_wave*, *sky_flux*, *sky_res*, *sky_ivar*, *outfile=None*, *pp=None*, *close=True*, *dpi=700*)

QA plot for residuals on sky lines ala Julien Guy :param sky_wave: :param sky_flux: :param sky_res: :param outfile: :param pp: :param close: :param nslices: :param dpi:

Returns:

desispec.qa.qa_plots.**skysub_gauss**(*sky_wave*, *sky_flux*, *sky_res*, *sky_ivar*, *outfile=None*, *pp=None*, *close=True*, *binsz=0.1*, *dpi=700*, *nfbin=4*)

Generate a plot examining the Gaussianity of the residuals Typically for a given channel :param wave: :param sky_flux: :param sky_res: :param sky_ivar: :param outfile: :param pp: :param close:

Returns:

desispec.qa.qa_plots.**skysub_resid_dual**(*sky_wave*, *sky_flux*, *sky_res*, *outfile=None*, *pp=None*, *close=True*, *nslices=20*, *dpi=700*)

Generate a plot of sky subtraction residuals Typically for a given channel :param wave: :param sky_flux: :param sky_res: :param outfile: :param pp: :param close:

Returns:

desispec.qa.qa_plots.**skysub_resid_series**(*sky_dict*, *xtype*, *outfile=None*, *pp=None*, *close=True*, *nslices=20*, *dpi=700*)

Generate a plot of sky subtraction residuals for a series of inputs Typically for a given channel :param wave: :param sky_flux: :param sky_res: :param outfile: :param pp: :param close:

Returns:

This includes routines to make pdf plots on the qa outputs from quicklook.

For information on QA dictionaries used here as input, visit wiki page: https://desi.lbl.gov/trac/wiki/Pipeline/QuickLook/QuicklookQAOutputs/Science

desispec.qa.qa_plots_ql.**plot_RMS**(*qa_dict*, *outfile*, *plotconf=None*, *hardplots=False*)

Plot RMS

Parameters

- **qa_dict** – dictionary of qa outputs from running qa_quicklook.Get_RMS
- **outfile** – Name of plot output file

desispec.qa.qa_plots_ql.**plot_SNR**(*qa_dict*, *outfile*, *objlist*, *fitsnr*, *rescut=0.2*, *sigmacut=2.0*, *plotconf=None*, *hardplots=False*)

Plot SNR

Parameters

- **qa_dict** – dictionary of qa outputs from running qa_quicklook.Calculate_SNR
- **outfile** – output png file
- **objlist** – list of objtype for log(snr**2) vs. mag plots
- **badfibs** – list of fibers with infs or nans to remove for plotting
- **fitsnr** – list of snr vs. mag fitting coefficients # JXP – THIS IS NOT TRUE!!

- **rescut** – only plot residuals (+/-) less than rescut (default 0.2)
- **sigmacut** – only plot residuals (+/-) less than sigma cut (default 2.0)
- **NOTE** – rescut taken as default cut parameter

desispec.qa.qa_plots_ql.**plot_XWSigma**(*qa_dict*, *outfile*, *plotconf=None*, *hardplots=False*)
Plot XWSigma

Parameters

- **qa_dict** – qa dictionary from countpix qa
- **outfile** – file of the plot

desispec.qa.qa_plots_ql.**plot_bias_overscan**(*qa_dict*, *outfile*, *plotconf=None*, *hardplots=False*)
Map of bias from overscan from 4 regions of CCD

Parameters

- **qa_dict** – qa dictionary from bias_from_overscan qa
- **outfile** – pdf file of the plot

desispec.qa.qa_plots_ql.**plot_countpix**(*qa_dict*, *outfile*, *plotconf=None*, *hardplots=False*)
Plot pixel counts above some threshold

Parameters

- **qa_dict** – qa dictionary from countpix qa
- **outfile** – pdf file of the plot

desispec.qa.qa_plots_ql.**plot_countspectralbins**(*qa_dict*, *outfile*, *plotconf=None*, *hardplots=False*)
Plot count spectral bins.

Parameters

- **qa_dict** – dictionary of qa outputs from running qa_quicklook.CountSpectralBins
- **outfile** – Name of figure.

desispec.qa.qa_plots_ql.**plot_lpolyhist**(*qa_dict*, *outfile*, *plotconf=None*, *hardplots=False*)
Plot histogram for each legendre polynomial coefficient in WSIGMA array.

Parameters

- **qa_dict** – Dictionary of qa outputs from running qa_quicklook.Check_Resolution
- **outfile** – Name of figure.

desispec.qa.qa_plots_ql.**plot_sky_continuum**(*qa_dict*, *outfile*, *plotconf=None*, *hardplots=False*)
Plot mean sky continuum from lower and higher wavelength range for each fiber and accross amps.

Parameters

- **qa_dict** – dictionary from sky continuum QA
- **outfile** – pdf file to save the plot

desispec.qa.qa_plots_ql.**plot_sky_peaks**(*qa_dict*, *outfile*, *plotconf=None*, *hardplots=False*)
Plot rms of sky peaks for smy fibers across amps

Parameters

- **qa_dict** – dictionary from sky peaks QA

- **outfile** – pdf file to save the plot

Class to organize QA for a full DESI production run

Monitoring algorithms for Quicklook pipeline

class desispec.qa.qa_quicklook.**Bias_From_Overscan**(*name*, *config*, *logger=None*)

class desispec.qa.qa_quicklook.**Calc_XWSigma**(*name*, *config*, *logger=None*)

class desispec.qa.qa_quicklook.**Calculate_SNR**(*name*, *config*, *logger=None*)

class desispec.qa.qa_quicklook.**Check_FiberFlat**(*name*, *config*, *logger=None*)

class desispec.qa.qa_quicklook.**Check_HDUs**(*name*, *config*, *logger=None*)

class desispec.qa.qa_quicklook.**Check_Resolution**(*name*, *config*, *logger=None*)

class desispec.qa.qa_quicklook.**CountSpectralBins**(*name*, *config*, *logger=None*)

class desispec.qa.qa_quicklook.**Count_Pixels**(*name*, *config*, *logger=None*)

class desispec.qa.qa_quicklook.**Get_RMS**(*name*, *config*, *logger=None*)

class desispec.qa.qa_quicklook.**Integrate_Spec**(*name*, *config*, *logger=None*)

class desispec.qa.qa_quicklook.**Sky_Continuum**(*name*, *config*, *logger=None*)

class desispec.qa.qa_quicklook.**Sky_Peaks**(*name*, *config*, *logger=None*)

class desispec.qa.qa_quicklook.**Sky_Rband**(*name*, *config*, *logger=None*)

class desispec.qa.qa_quicklook.**Sky_Residual**(*name*, *config*, *logger=None*)
Use offline sky_residual function to calculate sky residuals

class desispec.qa.qa_quicklook.**Trace_Shifts**(*name*, *config*, *logger=None*)

desispec.qa.qa_quicklook.**get_frame**(*filetype*, *night*, *expid*, *camera*, *specdir*)
Make frame object from file if in development mode

desispec.qa.qa_quicklook.**get_image**(*filetype*, *night*, *expid*, *camera*, *specdir*)
Make image object from file if in development mode

desispec.qa.qa_quicklook.**get_inputs**(**args*, ***kwargs*)
Get inputs required for each QA

simple low level library functions for offline and online qas

desispec.qa.qalib.**SN_ratio**(*flux*, *ivar*)
SN Ratio median snr for the spectra, flux should be sky subtracted.

Parameters

- **flux** (*array*) – 2d [nspec,nwave] the signal (typically for spectra, this comes from frame object
- **ivar** (*array*) – 2d [nspec,nwave] corresponding inverse variance

Returns 1d [nspec]

Return type medsnr (array)

desispec.qa.qalib.**SignalVsNoise**(*frame*, *params*, *fidboundary=None*)
Signal vs. Noise

Take flux and inverse variance arrays and calculate S/N for individual targets (ELG, LRG, QSO, STD) and for each amplifier of the camera.

Parameters

- **flux** (*array*) – 2d [nspec,nwave] the signal (typically for spectra, this comes from frame object
- **ivar** (*array*) – 2d [nspec,nwave] corresponding inverse variance
- **fidboundary** – list of slices indicating where to select in fiber and wavelength directions for each amp (output of slice_fidboundary function)

desispec.qa.qalib.**_get_mags**(*frame*)
Extract frame.fibermap fluxes into mags depending upon camera

Parameters **frame** – Frame object

Returns array of magnitudes, using 99.0 when flux<0

b camera frames return g-band magnitudes; r camera -> r-mags; z camera -> z-mags

desispec.qa.qalib.**ampregion**(*image*)
Get the pixel boundary regions for amps

Parameters **image** – desispec.image.Image object

desispec.qa.qalib.**continuum**(*wave*, *flux*, *wmin=None*, *wmax=None*)
Find the median continuum of the spectrum inside a wavelength region.

Parameters

- **wave** – 1d wavelength array
- **flux** – 1d counts/flux array
- **and wmax** (*wmin*) – region to consider for the continuum

desispec.qa.qalib.**countbins**(*flux*, *threshold=0*)
Count the number of bins above a given threshold on each fiber

Parameters

- **flux** – 2d (nspec,nwave)
- **threshold** – threshold counts

desispec.qa.qalib.**countpix**(*image*, *nsig=None*)
Count the pixels above a given threshold in units of sigma.

Parameters

- **image** – 2d image array
- **nsig** – threshold in units of sigma, e.g 2 for 2 sigma

desispec.qa.qalib.**fiducialregion**(*frame*, *psf*)
Get the fiducial amplifier regions on the CCD pixel to fiber by wavelength space

Parameters

- **frame** – desispec.frame.Frame object
- **psf** – desispec.psf.PSF like object

desispec.qa.qalib.**gauss**(*x*, *a*, *mu*, *sigma*)
Gaussian fit of input data

desispec.qa.qalib.**getrms**(*image*)
Calculate the rms of the pixel values)

Parameters **image** – 2d array

desispec.qa.qalib.**integrate_spec**(*wave*, *flux*)
Calculate the integral of the spectrum in the given range using trapezoidal integration

Note: limits of integration are min and max values of wavelength

Parameters

- **wave** – 1d wavelength array
- **flux** – 1d flux array

desispec.qa.qalib.**orig_SNRFit**(*frame*, *night*, *camera*, *expid*, *params*, *fidboundary=None*, *offline=False*)
Signal vs. Noise With fitting

Take flux and inverse variance arrays and calculate S/N for individual targets (ELG, LRG, QSO, STD) and for each amplifier of the camera. then fit the log(snr)=a+b*mag or log(snr)=poly(mag)

see http://arXiv.org/abs/0706.1062v2 for proper fitting of power-law distributions it is not implemented here!

qadict has the following data model "MAGNITUDES" : ndarray - Depends on camera (DECAM_G, DECAM_R, DECAM_Z) "MEDIAN_SNR" : ndarray (nfiber) "NUM_NEGATIVE_SNR" : int "SNR_MAG_TGT" "FITCOEFF_TGT" : list "SNR_RESID" : list, can be trimmed down during the fitting "FIDSNR_TGT" "RA" : ndarray (nfiber) "DEC" : ndarray (nfiber) "OBJLIST" : list - Save a copy to make sense of the list order later "EXPTIME" : float "FIT_FILTER" : str "r2" : float - Fitting parameter

Parameters

- **frame** – desispec.Frame object
- **night** –
- **camera** –
- **expid** – int
- **params** – parameters dictionary
- **{** – "Func": "linear", # Fit function type one of ["linear","poly","astro"] "FIDMAG": 22.0, # magnitude to evaluate the fit "Filter":"DECAM_R", #filter name
- **}** –
- **fidboundary** – list of slices indicating where to select in fiber and wavelength directions for each amp (output of slice_fidboundary function)
- **offline** – bool, optional If True, save things differently for offline

Returns dict

Return type qadict

desispec.qa.qalib.**s2n_flux_astro**(*flux*, *A*, *B*)
Function for a normalized (by texp**1/2) curve to flux vs S/N

Parameters

- **flux** (*float or np.ndarray*) – Flux value(s)
- **A** (*float*) – Scale coefficient
- **B** (*float*) – Offset coefficient

Returns S/N at the input flux

desispec.qa.qalib.**s2n_funcs**(*exptime=None*)
Functions for fitting S/N

Parameters **exptime** – float, optional

Returns dict

Return type funcMap

desispec.qa.qalib.**s2nfit**(*frame*, *camera*, *params*)
Signal vs. Noise With fitting

Take flux and inverse variance arrays and calculate S/N for individual targets (ELG, LRG, QSO, STD) and for each amplifier of the camera. then fit snr=A*mag/sqrt(A*mag+B)

see http://arXiv.org/abs/0706.1062v2 for proper fitting of power-law distributions it is not implemented here!

Instead we use scipy.optimize.curve_fit

Parameters

- **frame** – desispec.Frame object
- **camera** – str, name of the camera
- **params** – parameters dictionary for S/N

Returns

dict MEDIAN_SNR (ndarray, nfiber): Median S/N of light in each fiber FIT_FILTER (str): Filter used for the fluxes EXPTIME (float): Exposure time XXX_FIBERID (list): Fibers matching ELG, LRG, BGS, etc. SNR_MAG_TGT (list): List of lists with S/N and mag of ELG, LRG, BGS, etc. FITCOEFF_TGT (list): List of fitted coefficients. Junk fits have np.nan OBJLIST (list): List of object types analyzed (1 or more fiber)

Return type qadict

desispec.qa.qalib.**sky_continuum**(*frame*, *wrange1*, *wrange2*)
QA Algorithm for sky continuum.

To be called from desispec.sky.qa_skysub and desispec.qa.qa_quicklook.Sky_Continuum.run_qa

Parameters

- **frame** –
- **wrange1** –
- **wrange2** –

Returns skyfiber, contfiberlow, contfiberhigh, meancontfiber, skycont

desispec.qa.qalib.**sky_resid**(*param*, *frame*, *skymodel*, *quick_look=False*)
QA Algorithm for sky residual To be called from desispec.sky.qa_skysub and desispec.qa.qa_quicklook.Sky_residual.run_qa :param param: dict of QA parameters :param frame: desispec.Frame object after sky subtraction :param skymodel: desispec.SkyModel object

Returns a qa dictionary for sky resid

desispec.qa.qalib.**slice_fidboundary**(*frame*, *leftmax*, *rightmin*, *bottommax*, *topmin*)
leftmax,rightmin,bottommax,topmin - Indices in spec-wavelength space for different amps (e.g output from fiducialregion function) #- This could be merged to fiducialregion function

Returns (list): list of tuples of slices for spec- wavelength boundary for the amps.

Module for QA support

desispec.qa.utils.**get_skyres**(*cframes*, *sub_sky=False*, *flatten=True*)

Parameters

- **cframes** – str or list Single cframe or a list of them
- **sub_sky** – bool, optional Subtract the sky? This should probably not be done
- **flatten** – bool, optional Return a flat, 1D array for each variable
- **combine** – bool, optional combine the individual sky fibers? Median 'smash'

Returns ndarray flux : ndarray res : ndarray ivar : ndarray

Return type wave

1.8.40 desispec.qproc.io

I/O routines for qproc objects

desispec.qproc.io.**read_qframe**(*filename*, *nspec=None*, *skip_resolution=False*)
Reads a frame fits file and returns its data.

Parameters

- **filename** – path to a file
- **skip_resolution** – bool, option Speed up read time (>5x) by avoiding the Resolution matrix

Returns desispec.Frame object with attributes wave, flux, ivar, etc.

desispec.qproc.io.**write_qframe**(*outfile*, *qframe*, *header=None*, *fibermap=None*, *units=None*)
Write a frame fits file and returns path to file written.

Parameters

- **outfile** – full path to output file, or tuple (night, expid, channel)
- **qframe** – desispec.qproc.QFrame object with wave, flux, ivar...

Optional: header: astropy.io.fits.Header or dict to override frame.header fibermap: table to store as FIBERMAP HDU

Returns full filepath of output file that was written

Note: to create a QFrame object to pass into write_qframe, qframe = QFrame(wave, flux, ivar)

1.8.41 desispec.qproc.qextract

desispec.qproc.qextract.**qproc_boxcar_extraction**(*xytraceset*, *image*, *fibers=None*, *width=7*, *fibermap=None*, *save_sigma=True*)
Fast boxcar extraction of spectra from a preprocessed image and a trace set

Parameters

- **xytraceset** – DESI XYTraceSet object
- **image** – DESI preprocessed Image object

Optional: fibers : 1D np.array of int (default is all fibers, the first fiber is always = 0) width : extraction boxcar width, default is 7 fibermap : table

Returns QFrame object

desispec.qproc.qfiberflat.**qproc_apply_fiberflat**(*qframe*, *fiberflat*, *return_flat=False*)
Apply a fiber flat to a qframe.

Inputs: qframe: desispec.qproc.qframe.QFrame object which will be modified fiberflat: desispec.fiberflat.FiberFlat object with the flat to apply

Optional: return_flat : if True, returns the flat field that has been applied

Returns nothing or the flat that has been applied.

desispec.qproc.qfiberflat.**qproc_compute_fiberflat**(*qframe*, *niter_meanspec=4*, *nsig_clipping=3.0*, *spline_res_clipping=20.0*, *spline_res_flat=5.0*)
Fast estimation of fiberflat

1.8.42 desispec.qproc.qframe

Lightweight wrapper class for spectra extract with qextract, row-by-row extraction (boxcar or profile)

desispec.qproc.qsky.**qproc_sky_subtraction**(*qframe*, *return_skymodel=False*)
Fast sky subtraction directly applied to the input qframe.

Args: qframe : DESI QFrame object

Optional: return_skymodel returns the skymodel as an array of same shape as qframe.flux

1.8.43 desispec.quicklook

desispec.quicklook.arcprocess.**process_arc**(*frame*, *linelist=None*, *npoly=2*, *nbins=2*, *domain=None*)
frame: desispec.frame.Frame object, preumably resolution not evaluated. linelist: line list to fit npoly: polynomial order for sigma expansion nbins: no of bins for the half of the fitting window return: coefficients of the polynomial expansion

desispec.quicklook.arcprocess.**sigmas_from_arc**(*wave*, *flux*, *ivar*, *linelist*, *n=2*)
Gaussian fitting of listed arc lines and return corresponding sigmas in pixel units Args: linelist: list of lines (A) for which fit is to be done n: fit region half width (in bin units): n=2 bins => (2*n+1)=5 bins fitting window.

desispec.quicklook.arcprocess.**write_psffile**(*infile*, *wcoeffs*, *wcoeffs_wavemin*, *wcoeffs_wavemax*, *outfile*, *wavestepsize=None*)
extract psf file, add wcoeffs, and make a new psf file preserving the traces etc. psf module will load this

desispec.quicklook.palib Low level functions to be from top level PAs

desispec.quicklook.palib.**apply_flux_calibration**(*frame*, *fluxcalib*)
Apply flux calibration to sky subtracted qframe Use offline algorithm, but assume qframe object is input and that it is on native ccd wavelength grid Calibration vector is resampled to frame wavelength grid

frame: QFrame object fluxcalib: FluxCalib object

Modifies frame.flux and frame.ivar

desispec.quicklook.palib.**get_resolution**(*wave*, *nspec*, *tset*, *usesigma=False*)
Calculates approximate resolution values at given wavelengths in the format that can directly feed resolution data of desispec.frame.Frame object.

wave: wavelength array nsepc: no of spectra (int) tset: desispec.xytraceset like object usesigma: allows to use sigma from psf file for resolution computation.

returns : resolution data (nspec,nband,nwave); nband = 1 for usesigma = False, otherwise nband=21

desispec.quicklook.palib.**project**(*x1*, *x2*)
return a projection matrix so that arrays are related by linear interpolation x1: Array with one binning x2: new binning

Return Pr: x1= Pr.dot(x2) in the overlap region

desispec.quicklook.palib.**resample_spec**(*wave*, *flux*, *outwave*, *ivar=None*)
rebinning conserving S/N Algorithm is based on http://www.ast.cam.ac.uk/%7Erfc/vpfit10.2.pdf Appendix: B.1

Args: wave : original wavelength array (expected (but not limited) to be native CCD pixel wavelength grid outwave: new wavelength array: expected (but not limited) to be uniform binning flux : df/dx (Flux per A) sampled at x ivar : ivar in original binning. If not None, ivar in new binning is returned.

Note: Full resolution computation for resampling is expensive for quicklook.

desispec.interpolation.resample_flux using weights by ivar does not conserve total S/N. Tests with arc lines show much narrow spectral profile, thus not giving realistic psf resolutions This algorithm gives the same resolution as obtained for native CCD binning, i.e, resampling has insignificant effect. Details,plots in the arc processing note.

class desispec.quicklook.pas.**PipelineAlg**(*name*, *inptype*, *outtype*, *config*, *logger=None*)
Simple base class for Pipeline algorithms

get_default_config()
return a dictionary of 3-tuples, field 0 is the name of the parameter field 1 is the default value of the parameter field 2 is the comment for human readeable format. Field 2 can be used for QLF to dynamically setup the display

Pipeline Preprocessing algorithms for Quicklook

class desispec.quicklook.procalgs.**ApplyFiberFlat**(*name*, *config*, *logger=None*)
PA to Apply the fiberflat field to the given frame

class desispec.quicklook.procalgs.**ApplyFiberFlat_QL**(*name*, *config*, *logger=None*)
PA to Apply the fiberflat field (QL) to the given frame

class desispec.quicklook.procalgs.**ApplyFiberFlat_QP**(*name*, *config*, *logger=None*)
PA to Apply the fiberflat field (QP) to the given qframe

class desispec.quicklook.procalgs.**ApplyFluxCalibration**(*name*, *config*, *logger=None*)
PA to apply flux calibration to the given sframe

class desispec.quicklook.procalgs.**BoxcarExtract**(*name*, *config*, *logger=None*)

do_boxcar(*tset*, *outwave*, *boxwidth=2.5*, *nspec=500*, *maskFile=None*, *usesigma=False*, *quick_resolution=False*)
Extracts spectra row by row, given the centroids

Parameters

- **image** – desispec.image object
- **tset** – desispec.xytraceset like object
- **outwave** – wavelength array for the final spectra output
- **boxwidth** – HW box size in pixels

- **usesigma** – if True, use sigma from psf file (ysigma) to calculate resolution data.
- **quick_resolution** – whether to calculate the resolution matrix or use QuickResolution object

Returns flux, ivar, resolution

get_default_config()
return a dictionary of 3-tuples, field 0 is the name of the parameter field 1 is the default value of the parameter field 2 is the comment for human readeable format. Field 2 can be used for QLF to dynamically setup the display

class desispec.quicklook.procalgs.**ComputeFiberflat**(*name*, *config*, *logger=None*)
PA to compute fiberflat field correction from a DESI continuum lamp frame

class desispec.quicklook.procalgs.**ComputeFiberflat_QL**(*name*, *config*, *logger=None*)
PA to compute fiberflat field correction from a DESI continuum lamp frame

class desispec.quicklook.procalgs.**ComputeFiberflat_QP**(*name*, *config*, *logger=None*)

class desispec.quicklook.procalgs.**ComputeSky**(*name*, *config*, *logger=None*)
PA to compute sky model from a DESI frame

class desispec.quicklook.procalgs.**ComputeSky_QL**(*name*, *config*, *logger=None*)
PA to compute sky model from a DESI frame

class desispec.quicklook.procalgs.**Extract_QP**(*name*, *config*, *logger=None*)

get_default_config()
return a dictionary of 3-tuples, field 0 is the name of the parameter field 1 is the default value of the parameter field 2 is the comment for human readeable format. Field 2 can be used for QLF to dynamically setup the display

class desispec.quicklook.procalgs.**Extraction_2d**(*name*, *config*, *logger=None*)
Offline 2D extraction for offline QuickLook

class desispec.quicklook.procalgs.**Flexure**(*name*, *config*, *logger=None*)
Use desi_compute_trace_shifts to output modified psf file

class desispec.quicklook.procalgs.**Initialize**(*name*, *config*, *logger=None*)
This PA takes information from the fibermap and raw header and adds it to the general info section of the merged dictionary

class desispec.quicklook.procalgs.**Preproc**(*name*, *config*, *logger=None*)

class desispec.quicklook.procalgs.**ResolutionFit**(*name*, *config*, *logger=None*)
Fitting of Arc lines on extracted arc spectra, polynomial expansion of the fitted sigmas, and updating the coefficients to the new traceset file

class desispec.quicklook.procalgs.**SkySub**(*name*, *config*, *logger=None*)

class desispec.quicklook.procalgs.**SkySub_QL**(*name*, *config*, *logger=None*)
This is for QL Sky subtraction. The input frame object should be fiber flat corrected. Unlike offline, if no skymodel file is given as input, a sky compute method is called to create a skymodel object and then subtraction is performed. Outputing that skymodel to a file is optional and can be configured.

class desispec.quicklook.procalgs.**SkySub_QP**(*name*, *config*, *logger=None*)
Sky subtraction. The input frame object should be fiber flat corrected. No sky model is saved for now

class desispec.quicklook.qas.**MonitoringAlg**(*name*, *inptype*, *config*, *logger=None*)
Simple base class for monitoring algorithms

class desispec.quicklook.qas.**QASeverity**
An enumeration.

boxcar extraction for Spectra from Desi Image

desispec.quicklook.qlboxcar.**do_boxcar**(*image*, *tset*, *outwave*, *boxwidth=2.5*, *nspec=500*, *maskFile=None*, *usesigma=False*, *quick_resolution=False*)
Extracts spectra row by row, given the centroids

Parameters

- **image** – desispec.image object
- **tset** – desispec.xytraceset like object
- **outwave** – wavelength array for the final spectra output
- **boxwidth** – HW box size in pixels
- **usesigma** – if True, use sigma from psf file (ysigma) to calculate resolution data.
- **quick_resolution** – whether to calculate the resolution matrix or use QuickResolution object

Returns flux, ivar, resolution

class desispec.quicklook.qlconfig.**Config**(*configfile*, *night*, *camera*, *expid*, *singqa*, *amps=True*, *rawdata_dir=None*, *specprod_dir=None*, *outdir=None*, *qlf=False*, *psfid=None*, *flatid=None*, *templateid=None*, *templatenight=None*, *qlplots=False*, *store_res=None*)
A class to generate Quicklook configurations for a given desi exposure. expand_config will expand out to full format as needed by quicklook.setup

dump_pa(*paname*)
dump the PA outputs to respective files. This has to be updated for fframe and sframe files as QL anticipates for dumpintermediate case.

dump_qa()
yaml outputfile for the set of qas for a given pa Name and default locations of files are handled by desispec.io.meta.findfile

expand_config()
config: desispec.quicklook.qlconfig.Config object

io_qa(*qaname*)
Specify the filenames: json and png for the given qa output

io_qa_pa(*paname*)
Specify the filenames: json and png of the pa level qa files"

paargs
Many arguments for the PAs are taken default. Some of these may need to be variable psfspfile is for offline extraction case

palist
palist for this config see :class: *Palist* for details.

qalist
qalist for the given palist

class desispec.quicklook.qlconfig.**Palist**(*thislist=None*, *algorithms=None*)
Generate PA list and QA list for the Quicklook Pipeline for the given exposure

desispec.quicklook.qlconfig.**check_config**(*outconfig*, *singqa*)
Given the expanded config, check for all possible file existence etc. . . .

exception desispec.quicklook.qlexceptions.**ParameterException**(*value*)

class desispec.quicklook.qllogger.**QLLogger**(*name=None*, *loglevel=20*)
Simple logger class using logging

Generic plotting algorithms for QuickLook QAs

desispec.quicklook.ql_plotlib.**ql_2dplot**(*ax*, *xvals*, *yvals*, *plottitle*, *xtitle*, *ytitle*, *xlim=None*, *ylim=None*)
Make 2d plot of specific metrics provided in configuration file

Parameters

- **ax** – matplotlib subplot
- **xvals** – QA metric to be plotted along the xaxis
- **yvals** – QA metric to be plotted along the yaxis
- **plottitle** – plot title from configuration file
- **xtitle** – x axis label
- **ytitle** – y axis label

Optional: xlim: list containing x range (i.e. [x_lo,x_hi]) ylim: list containing y range (i.e. [y_lo,y_hi])

Returns matplotlib sublot containing plotted metrics

desispec.quicklook.ql_plotlib.**ql_3dplot**(*ax*, *xvals*, *yvals*, *zvals*, *plottitle*, *xtitle*, *ytitle*, *zlim=None*, *heatmap=None*)
Make 3d scatter plot of specific metrics provided in configuration file

Parameters

- **ax** – matplotlib subplot
- **xvals** – QA metric to be plotted along the xaxis
- **yvals** – QA metric to be plotted along the yaxis
- **zvals** – QA metric to be plotted
- **plottitle** – plot title from configuration file
- **xtitle** – x axis label
- **ytitle** – y axis label

Optional: zlim: list containing scatter plot range (i.e. [z_lo,z_hi])

Returns matplotlib sublot containing plotted metrics

desispec.quicklook.ql_plotlib.**ql_patchplot**(*ax*, *vals*, *plottitle*, *grid*, *heatmap=None*)
Make patch plot of specific metrics provided in configuration file

Parameters

- **ax** – matplotlib subplot

- **vals** – QA metric to be plotted
- **plottitle** – plot title from configuration file
- **grid** – shape of patch plot

Optional: heat: specify color of heatmap (must conform to matplotlib)

Returns matplotlib sublot containing plotted metrics

desispec.quicklook.ql_plotlib.**ql_qaplot**(*fig*, *plotconf*, *qadict*, *camera*, *expid*, *outfile*)
Get plotting configuration info and setup plots

Parameters

- **fig** – matplotlib figure
- **plotconf** – list of config info for each plot
- **qadict** – QA metrics dictionary
- **expid** (*camera,*) – to be used in output png title
- **outfile** – output png file

Returns png file containing all desired plots

Given a psf output file eg. output from bootcalib.write_psf or desimodel/data/specpsf/PSF* files this defines an interface that other codes can use the trace and wavelength solutions

Mostly making parallel to specter.psf.PSF baseclass and inheriting as needed, but only xtrace, ytrace and wavelength solution available for this case. No resolution information yet.

class desispec.quicklook.qlpsf.**PSF**(*filename*)
Base class for 2D psf

angstroms_per_pixel(*ispec*, *wavelength*)
Return CCD pixel width in Angstroms for spectrum ispec at given wavlength(s). Wavelength may be scalar or array.

wavelength(*ispec=None*, *y=None*)
returns wavelength evaluated at y

x(*ispec=None*, *wavelength=None*)
returns CCD x centroids for the spectra ispec can be None, scalar or a vector wavelength can be None or a vector

y(*ispec=None*, *wavelength=None*)
returns CCD y centroids for the spectra ispec can be None, scalar or a vector wavelength can be a vector but not allowing None #- similar as in specter.psf.PSF.y

1.8.44 desispec.quicklook.qlresolution

Quicklook version of resolution object that can calculate resolution efficiently from psf information

Author: Sami Kama

class desispec.quicklook.qlresolution.**QuickResolution**(*mu=None*, *sigma=None*, *wdict=None*, *waves=None*, *ndiag=9*)
Quicklook version of the resolution mimicking desispec.resolution.Resolution with some reduction in

dimentionality. Contains code from Resolution implementation Note that this is similar to desispec.resolution.Resolution, though faster and differing in implementation details that should be cross checked before merging these or replacing one with the other

desispec.quickfiberflat

Here will be the fiberflat routines specific to quicklook.

G. Dhungana, 2016

desispec.quicklook.quickfiberflat.**apply_fiberflat**(*frame*, *fiberflat*)

Args: frame: desispec.frame.Frame object fiberflat: desispec.fiberflat.Fiberflat object

desispec.quicklook.quickfiberflat.**compute_fiberflat**()
computes fiberflat: A boss like algorithm writing in progress and will fit in here.

Args:

desispec.quicklook.quicklook.**get_chan_spec_exp**(*inpname*, *camera=None*)
Get channel, spectrograph and expid from the filename itself

Parameters

- **inpname** – can be raw or pix, or frame etc filename
- **camera** – is required for raw case, eg, r0, b5, z8 irrelevant for others

desispec.quicklook.quicklook.**mapkeywords**(*kw*, *kwmap*)
Maps the keyword in the configuration to the corresponding object returned by the desispec.io module. e.g Bias Image file is mapped to biasimage object... for the same keyword "BiasImage"

desispec.quicklook.quicklook.**runpipeline**(*pl*, *convdict*, *conf*)
Runs the quicklook pipeline as configured

Parameters

- **pl** – is a list of [pa,qas] where pa is a pipeline step and qas the corresponding qas for that pa
- **convdict** – converted dictionary e.g : conf["IMAGE"] is the real psf file but convdict["IMAGE"] is like desispec.image.Image object and so on. details in setup_pipeline method below for examples.
- **conf** – a configured dictionary, read from the configuration yaml file. e.g: conf=configdict=yaml.safe_load(open('configfile.yaml','rb'))

desispec.quicklook.quicklook.**setup_pipeline**(*config*)
Given a configuration from QLF, this sets up a pipeline [pa,qa] and also returns a conversion dictionary from the configuration dictionary so that Pipeline steps (PA) can take them. This is required for runpipeline.

desispec.quicklook.quicksky

Here will be the sky computing and sky subtraction routines for QL

desispec.quicklook.quicksky.**compute_sky**(*fframe*, *fibermap=None*, *nsig_clipping=4.0*, *apply_resolution=False*)
Adding in the offline algorithm here to be able to apply resolution for sky compute. We will update this here as needed for quicklook. The original weighted sky compute still is the default.

Args: fframe: fiberflat fielded frame object fibermap: fibermap object apply_resolution: if True, uses the resolution in the frame object to evaluate sky allowing fiber to fiber variation of resolution.

desispec.quicklook.quicksky.**subtract_sky**(*fframe*, *skymodel*)
skymodel: skymodel object. fframe: frame object to do the sky subtraction, should be already fiber flat fielded need same number of fibers and same wavelength grid

1.8.45 desispec.resolution

Standardized handling of sparse wavelength resolution matrices.

Use `python -m desispec.resolution` to run unit tests.

class desispec.resolution.**Resolution**(*data*, *offsets=None*)
Canonical representation of a resolution matrix.

Inherits all of the method of scipy.sparse.dia_matrix, including todense() for converting to a dense 2D numpy array of matrix elements, most of which will be zero (so you generally want to avoid this).

Parameters **data** – Must be in one of the following formats listed below.

Options:

offsets: list of diagonals that the data represents. Only used if data is a 2D dense array.

Raises ValueError – Invalid input for initializing a sparse resolution matrix.

Data formats:

1. a scipy.sparse matrix in DIA format with the required diagonals (but not necessarily in the canoncial order);
2. a 2D square numpy arrray (i.e., a dense matrix) whose non-zero values beyond default_ndiag will be silently dropped; or
3. a 2D numpy array[ndiag, nwave] that encodes the sparse diagonal values in the same format as scipy.sparse.dia_matrix.data .

The last format is the one used to store resolution matrices in FITS files.

to_fits_array()
Convert to an array of sparse diagonal values.

This is the format used to store resolution matrices in FITS files. Note that some values in the returned rectangular array do not correspond to actual matrix elements since the diagonals get smaller as you move away from the central diagonal. As long as you treat this array as an opaque representation for FITS I/O, you don't care about this. To actually use the matrix, create a Resolution object from the fits array first.

Returns

An array of (num_diagonals,nbins) sparse matrix element values close to the diagonal.

Return type numpy.ndarray

desispec.resolution.**_gauss_pix**(*x*, *mean=0.0*, *sigma=1.0*)
Utility function to integrate Gaussian density within pixels

Parameters

- **x** (*1D array*) – pixel centers
- **mean** (*float*) – mean of Gaussian
- **sigma** (*float*) – sigma of Gaussian

Returns array of integals of the Gaussian density in the pixels.

Note: All pixels must be the same size

---

desispec.resolution.**_sort_and_symmeterize**(*data*, *offsets*)
Sort data,offsets and pad to ensure equal number of upper/lower diagonals

Parameters

- **data** – 2D array of diagonals, following scipy.sparse.dia_matrix.data ordering
- **offsets** – 1D array of offsets; must be complete from min to max but doesn't have to be sorted

Returns fulldata, fulloffsets

1.8.46 desispec.scripts

Main functions and commandline parsing.

1.8.47 desispec.scripts.average_fiberflat

1.8.48 desispec.scripts.bootcalib

Utility functions to perform a quick calibration of DESI data

TODO: 1. Expand to r, i cameras 2. QA plots 3. Test with CR data Extract spectra from DESI pre-processed raw data

desispec.scripts.extract.**_extract_and_save**(*img*, *psf*, *bspecmin*, *bnspec*, *specmin*, *wave*, *raw_wave*, *fibers*, *fibermap*, *outbundle*, *outmodel*, *bundlesize*, *args*, *log*)

Performs the main extraction and saving of extracted frames found in the body of the main loop. Refactored to be callable by both MPI and non-MPI versions of the code. This should be viewed as a shorthand for the following commands.

desispec.scripts.extract.**barycentric_correction_multiplicative_factor**(*header*)

Returns mult. barycentric correction factor using coords in *header*

header must contrain MJD or MJD-OBS; and TARGTRA,TARGTDEC or SKYRA,SKYDEC or TELRA,TELDEC or RA,DEC

desispec.scripts.extract.**gpu_specter_check_input_options**(*args*)

Perform pre-flight checks on input options

returns ok(True/False), message

1.8.49 desispec.scripts.fiberflat

Utility functions to compute a fiber flat correction and apply it We try to keep all the (fits) io separated. Regroup spectra by healpix

exspec extracts individual bundles of spectra with one bundle per output file. This script merges them back together into a single file combining all bundles.

This workflow is hacky. Release early, release often, but also refactor often.

Stephen Bailey, LBL March 2014

1.8.50 desispec.scripts.night

Automated nightly processing. Interactive control of the pipeline

Run one or more pipeline tasks.

1.8.51 desispec.scripts.preproc

Command line wrappers for pre-processing a DESI raw exposure

desispec.scripts.preproc.**_preproc_file_kwargs_wrapper**(*opts*)
This function just unpacks opts dict for preproc_file so that it can be used with multiprocessing.Pool.map

desispec.scripts.preproc.**preproc_file**(*infile*, *camera*, *outfile=None*, *outdir=None*, *fibermap=None*, *zero_masked=False*, ***preproc_opts*)
Preprocess a single camera from a single input file

Parameters

- **infile** – input raw data file
- **camera** – camera, e.g. 'b0', 'r1', 'z9'

Options: outfile: output preprocessed image file to write outdir: output directory; derive filename from infile NIGHT and EXPID fibermap: fibermap filename to include in output zero_masked (bool): set masked pixels to 0 preproc_opts: dictionary to pass to preproc

Returns error code (1=error, 0=success) but will not raise exception if there is an I/O or preprocessing failure (allows other parallel procs to proceed).

Note: either *outfile* or *outdir* must be provided

This script processes an exposure by applying fiberflat, sky subtraction, spectro-photometric calibration depending on input. Optionally, includes tsnr in the scores hdu.

1.8.52 desispec.scripts.quicklook

Command line wrapper for running a QL pipeline

QuickLook team @Southern Methodist University (SMU) First version Spring 2016 Latest revision July 2018

Running QuickLook:

desi_quicklook -i qlconfig_science.yaml -n 20191001 -c r0 -e 3577

This requires having necessary input files and setting the following environment variables:

QL_SPEC_DATA: directory containing raw/fibermap files (full path: $QL_SPEC_DATA/night/expid) QL_SPEC_REDUX: directory for QL output (full path: $QL_SPEC_REDUX/exposures/night/expid) DESI_CALIBRATION_DATA: directory containing calibration files

Necessary Quicklook command line arguments:

-i,–config_file : path to QL configuration file -n,–night : night to be processed -c,–camera : camera to be processed -e,–expid : exposure ID to be processed

Optional QuickLook arguments:

–rawdata_dir : directory containing raw/fibermap files (overrides $QL_SPEC_DATA) –specprod_dir : directory for QL output (overrides $QL_SPEC_REDUX)

Plotting options:

-p (including path to plotting configuration file) : generate configured plots -p (only using -p with no configuration file) : generate QL hardcoded plots

desispec.scripts.quicklook.**parse**()
Should have either a pre existing config file, or need to generate one using config module

This script finds cosmics in a pre-processed image and write the result in the mask extension of an output image (output can be same as input).

Run PSF estimation.

desispec.scripts.specex.**compatible**(*head1*, *head2*)
Return bool for whether two FITS headers are compatible for merging PSFs

desispec.scripts.specex.**mean_psf**(*inputs*, *output*)
Average multiple input PSF files into an output PSF file

Parameters

- **inputs** – list of input PSF files
- **output** – output filename

desispec.scripts.specex.**merge_psf**(*inputs*, *output*)
Merge individual per-bundle PSF files into full PSF

Parameters

- **inputs** – list of input PSF filenames
- **output** – output filename

desispec.scripts.specex.**run**(*comm*, *cmds*, *cameras*)
Run PSF fits with specex on a set of ccd images in parallel using the run method of the desispec.workflow.schedule.Schedule (Schedule) class.

Parameters

- **comm** – MPI communicator containing all processes available for work and scheduling (usually MPI_COMM_WORLD); at least 21 processes should be available, one for scheduling and (group_size=) 20 to fit all bundles for a given ccd image. Otherwise there is no constraint on the number of ranks available, but (comm.Get_size()-1)%group_size will be unused, since every job is assigned exactly group_size=20 ranks. The variable group_size is set at the number of bundles on a ccd, and there is currently no support for any other number, due to the way merging of bundles is currently done.
- **cmds** – dictionary keyed by a camera string (e.g. 'b0', 'r1', ...) with values being the 'desi_compute_psf ...' string that one would run on the command line.
- **cameras** – list of camera strings identifying the entries in cmds to be run as jobs in parallel jobs, one entry per ccd image to be fit; entries not in the dictionary will be logged as an error while still continuing with the others and not crashing.

The function first defines the procedure to call specex for a given ccd image with the "fitbundles" inline function, passes the fitbundles function to the Schedule initialization method, and then calls the run method of the Schedule class to call fitbundles len(cameras) times, each with group_size = 20 processes.

Compute some information scores on spectra in frames

Get the normalized best template to do flux calibration.

desispec.scripts.stdstars.**get_gaia_ab_correction**()
Get the dictionary with corrections from AB magnitudes to Vega magnitudes (as the official gaia catalog is in vega)

desispec.scripts.stdstars.**get_magnitude**(*stdwave*, *model*, *model_filters*, *cur_filt*)
Obtain magnitude for a filter taking into account the ab/vega correction if needed. Wwe assume the flux is in units of 1e-17 erg/s/cm^2/A

desispec.scripts.stdstars.**main**(*args*, *comm=None*)
finds the best models of all standard stars in the frame and normlize the model flux. Output is written to a file and will be called for calibration.

1.8.53 desispec.scripts.trace_shifts

Update healpix-grouped spectra from a set of input cframe files

1.8.54 desispec.sky

Utility functions to compute a sky model and subtract it.

desispec.sky.**_model_variance**(*frame*, *cskyflux*, *cskyivar*, *skyfibers*)
look at chi2 per wavelength and increase sky variance to reach chi2/ndf=1

desispec.sky.**calculate_throughput_corrections**(*frame*, *skymodel*)
Calculate the throughput corrections for each fiber based on the skymodel.

Parameters

- **frame** (*Frame object*) – frame containing the data that may need to be corrected
- **skymodel** (*SkyModel object*) – skymodel object that contains the information about the sky for the given exposure/frame

Output:

corrections (1D array): 1D array where the index corresponds to the fiber % 500 and the values are the multiplicativ
be applied to the fluxes in frame.flux to correct them based on the input skymodel

desispec.sky.**compute_non_uniform_sky**(*frame*, *nsig_clipping=4.0*, *max_iterations=10*, *model_ivar=False*, *add_variance=True*, *angular_variation_deg=1*)
Compute a sky model.

Sky[fiber,i] = R[fiber,i,j] (Flux_0[j] + x[fiber]*Flux_x[j] + y[fiber]*Flux_y[j] + ...)

Input flux are expected to be flatfielded! We don't check this in this routine.

Parameters

- **frame** – Frame object, which includes attributes - wave : 1D wavelength grid in Angstroms - flux : 2D flux[nspec, nwave] density - ivar : 2D inverse variance of flux - mask : 2D inverse mask flux (0=good) - resolution_data : 3D[nspec, ndiag, nwave] (only sky fibers)
- **nsig_clipping** – [optional] sigma clipping value for outlier rejection

Optional: max_iterations : int , number of iterations model_ivar : replace ivar by a model to avoid bias due to correlated flux and ivar. this has a negligible effect on sims. add_variance : evaluate calibration error and add this to the sky model variance angular_variation_deg : degree of 2D polynomial correction as a function of fiber focal plane coordinates (default=1). One set of coefficients per wavelength

returns SkyModel object with attributes wave, flux, ivar, mask

desispec.sky.**compute_polynomial_times_sky**(*frame*, *nsig_clipping=4.0*, *max_iterations=30*, *model_ivar=False*, *add_variance=True*, *angular_variation_deg=1*, *chromatic_variation_deg=1*)
Compute a sky model.

Sky[fiber,i] = R[fiber,i,j] Polynomial(x[fiber],y[fiber],wavelength[j]) Flux[j]

Input flux are expected to be flatfielded! We don't check this in this routine.

Parameters

- **frame** – Frame object, which includes attributes - wave : 1D wavelength grid in Angstroms - flux : 2D flux[nspec, nwave] density - ivar : 2D inverse variance of flux - mask : 2D inverse mask flux (0=good) - resolution_data : 3D[nspec, ndiag, nwave] (only sky fibers)
- **nsig_clipping** – [optional] sigma clipping value for outlier rejection

Optional: max_iterations : int , number of iterations model_ivar : replace ivar by a model to avoid bias due to correlated flux and ivar. this has a negligible effect on sims. add_variance : evaluate calibration error and add this to the sky model variance

returns SkyModel object with attributes wave, flux, ivar, mask

desispec.sky.**compute_sky**(*frame*, *nsig_clipping=4.0*, *max_iterations=100*, *model_ivar=False*, *add_variance=True*, *angular_variation_deg=0*, *chromatic_variation_deg=0*, *adjust_wavelength=False*, *adjust_lsf=False*, *only_use_skyfibers_for_adjustments=True*, *pcacorr=None*, *fit_offsets=False*, *fiberflat=None*)

Compute a sky model.

Input flux are expected to be flatfielded! We don't check this in this routine.

Parameters

- **frame** – Frame object, which includes attributes - wave : 1D wavelength grid in Angstroms - flux : 2D flux[nspec, nwave] density - ivar : 2D inverse variance of flux - mask : 2D inverse mask flux (0=good) - resolution_data : 3D[nspec, ndiag, nwave] (only sky fibers)
- **nsig_clipping** – [optional] sigma clipping value for outlier rejection

Optional: max_iterations : int , number of iterations model_ivar : replace ivar by a model to avoid bias due to correlated flux and ivar. this has a negligible effect on sims. add_variance : evaluate calibration error and add this to the sky model variance angular_variation_deg : Degree of polynomial for sky flux variation with focal plane coordinates (default=0, i.e. no correction, a uniform sky) chromatic_variation_deg : Wavelength degree for the chromatic x angular terms. If negative, use as many 2D polynomials of x and y as wavelength entries. adjust_wavelength : adjust the wavelength of the sky model on sky lines to improve the sky subtraction adjust_lsf : adjust the LSF width of the sky model on sky lines to improve the sky subtraction only_use_skyfibers_for_adjustments: interpolate adjustments using sky fibers only pcacorr : SkyCorrPCA object to interpolate the wavelength or LSF adjustment from sky fibers to all fibers fit_offsets : fit offsets for regions defined in calib fiberflat : desispec.FiberFlat object used for the fit of offsets

returns SkyModel object with attributes wave, flux, ivar, mask

desispec.sky.**compute_uniform_sky**(*frame*, *nsig_clipping=4.0*, *max_iterations=100*, *model_ivar=False*, *add_variance=True*, *adjust_wavelength=True*, *adjust_lsf=True*, *only_use_skyfibers_for_adjustments=True*, *pcacorr=None*, *fit_offsets=False*, *fiberflat=None*)

Compute a sky model.

Sky[fiber,i] = R[fiber,i,j] Flux[j]

Input flux are expected to be flatfielded! We don't check this in this routine.

Parameters

- **frame** – Frame object, which includes attributes - wave : 1D wavelength grid in Angstroms - flux : 2D flux[nspec, nwave] density - ivar : 2D inverse variance of flux - mask : 2D inverse mask flux (0=good) - resolution_data : 3D[nspec, ndiag, nwave] (only sky fibers)
- **nsig_clipping** – [optional] sigma clipping value for outlier rejection

Optional: max_iterations : int , number of iterations model_ivar : replace ivar by a model to avoid bias due to correlated flux and ivar. this has a negligible effect on sims. add_variance : evaluate calibration error and add this to the sky model variance adjust_wavelength : adjust the wavelength of the sky model on sky lines to improve the sky subtraction adjust_lsf : adjust the LSF width of the sky model on sky lines to improve the sky subtraction only_use_skyfibers_for_adjustments : interpolate adjustments using sky fibers only pcacorr : SkyCorrPCA object to interpolate the wavelength or LSF adjustment from sky fibers to all fibers fit_offsets : fit offsets for regions defined in calib fiberflat : desispec.FiberFlat object used for the fit of offsets

returns SkyModel object with attributes wave, flux, ivar, mask

desispec.sky.**qa_skysub**(*param*, *frame*, *skymodel*, *quick_look=False*)
Calculate QA on SkySubtraction

Note: Pixels rejected in generating the SkyModel (as above), are not rejected in the stats calculated here. Would need to carry along current_ivar to do so.

Parameters

- **param** – dict of QA parameters : see qa_frame.init_skysub for example
- **frame** – desispec.Frame object; Should have been flat fielded
- **skymodel** – desispec.SkyModel object
- **quick_look** – bool, optional If True, do QuickLook specific QA (or avoid some)

Returns

dict of QA outputs Need to record simple Python objects for yaml (str, float, int)

Return type qadict

desispec.sky.**subtract_sky**(*frame*, *skymodel*, *apply_throughput_correction=True*, *zero_ivar=True*)
Subtract skymodel from frame, altering frame.flux, .ivar, and .mask

Parameters

- **frame** – desispec.Frame object
- **skymodel** – desispec.SkyModel object

Option:

apply_throughput_correction [if True, fit for an achromatic throughput correction.] This is to absorb variations of Focal Ratio Degradation with fiber flexure.

zero_ivar : if True , set ivar=0 for masked pixels

1.8.55 desispec.specscore

Spectral scores routines.

desispec.specscore.**compute_coadd_scores**(*coadd*, *specscores=None*, *update_coadd=True*)
Compute scores for a coadded Spectra object

Parameters **coadd** – a Spectra object from a coadd

Options: update_coadd: if True, update coadd.scores specscores: scores Table from the uncoadded spectra including a TARGETID column

Returns tuple of dictionaries (scores, comments); see compute_frame_scores

`specscores` is used to update TSNR2 scores by summing inputs

`desispec.specscore.`**compute_coadd_tsnr_scores**(*specscores*)
Compute coadded TSNR2 scores (TSNR2=Template Signal-to-Noise squared)

Parameters **specscores** – uncoadded scores with TSNR2* columns (dict or Table-like)

Returns (tsnrscores, comments) tuple of dictionaries

`desispec.specscore.`**compute_frame_scores**(*frame*, *band=None*, *suffix=None*, *flux_per_angstrom=None*)
Computes scores in spectra of a frame.

The scores are sum,mean,medians in a predefined and fixed wavelength range for each DESI camera arm, or band, b, r or z. The band argument is optional because it can be automatically chosen from the wavelength range in the frame. The suffix is added to the key name in the output dictionnary, for instance 'RAW', 'SKYSUB', 'CALIB' ... The boolean argument flux_per_angstrom is needed if there is no 'BUNIT' keyword in frame.meta (frame fits header)

Parameters

- **frame** (`Frame` or `QFrame`) – A Frame or a QFrame object.
- **band** (`str`, optional) – Spectrograph band, `b`, `r`, `z`, autodetected by default.
- **suffix** (`str`, optional) – Character string added to the keywords in the output dictionary, for instance suffix='RAW'
- **flux_per_angstrom** (`bool`, optional) – If `True` the spectra are assumed flux_per_angstrom, *i.e.* flux densities. If `False`, the spectra are assumed to be counts or photo-electrons per bin. `None` by default in which case the `frame.units` string is read to find out whether the flux quantity is per unit wavelenght or per bin.

Returns A tuple containg a `dict` of 1D arrays of size = number of spectra in frame and a `dict` of string with comments on the type of scores.

Return type `tuple()`

1.8.56 desispec.spectra

Class for dealing with a group of spectra from multiple bands and the associated fibermap information.

class `desispec.spectra.`**Spectra**(*bands=[]*, *wave={}*, *flux={}*, *ivar={}*, *mask=None*, *resolution_data=None*, *fibermap=None*, *exp_fibermap=None*, *meta=None*, *extra=None*, *single=False*, *scores=None*, *scores_comments=None*, *extra_catalog=None*)
Represents a grouping of spectra.

This class contains an "extended" fibermap that has information about the night and exposure of each spectrum. For each band, this class has the wavelength grid, flux, ivar, mask, and resolution arrays.

Parameters

- **bands** (`list`) – List of strings used to identify the bands.

- **wave** (dict) – Dictionary of arrays specifying the wavelength grid.
- **flux** (dict) – Dictionary of arrays specifying the flux for each spectrum.
- **ivar** (dict) – Dictionary of arrays specifying the inverse variance.
- **mask** (dict, optional) – Dictionary of arrays specifying the bitmask.
- **resolution_data** (dict, optional) – Dictionary of arrays specifying the block diagonal resolution matrix. The object for each band must be in one of the formats supported by the Resolution class constructor.
- **Table-like, optional** (*exp_fibermap,*) – Extended fibermap to use. If not specified, a fake one is created.
- **Table-like, optional** – Exposure-specific fibermap columns, which may not apply to a coadd.
- **meta** (dict, optional) – Dictionary of arbitrary properties.
- **extra** (dict, optional) – Optional dictionary of dictionaries containing extra floating point arrays. The top-level is a dictionary over bands and each value is a dictionary containing string keys and values which are arrays of the same size as the flux array.
- **single** (bool, optional) – If True, store data in memory as single precision.
- **scores** – QA scores table.
- **scores_comments** – dict[column] = comment to include in output file
- **extra_catalog** (*numpy or astropy Table, optional*) – optional table of metadata, rowmatched to fibermap, e.g. a redshift catalog for these spectra

_get_slice(*index*, *bands=None*)
Slice spectra by index. :param bands: optional list of bands to select. :type bands: list

Note: This function is intended to be private, to be used by __getitem__() and select().

bands
the list of valid bands.

Type (list)

ftype
the data type used for floating point numbers.

Type (numpy.dtype)

num_spectra()
Get the number of spectra contained in this group.

Returns (int): Number of spectra contained in this group.

num_targets()
Get the number of distinct targets.

Returns (int): Number of unique targets with spectra in this object.

select(*nights=None*, *exposures=None*, *bands=None*, *targets=None*, *fibers=None*, *invert=False*, *return_index=False*)
Select a subset of the data.

This filters the data based on a logical AND of the different criteria, optionally inverting that selection.

Parameters

- **nights** (*list*) – optional list of nights to select.

- **exposures** (*list*) – optional list of exposures to select.
- **bands** (*list*) – optional list of bands to select.
- **targets** (*list*) – optional list of target IDs to select.
- **fibers** (*list*) – list/array of fiber indices to select.
- **invert** (*bool*) – after combining all criteria, invert selection.
- **return_index** (*bool*) – if True, also return the indices of selected spectra.

Returns a new Spectra object containing the selected data. indices (list, optional): indices of selected spectra. Only provided if return_index is True.

Return type spectra

target_ids()
Return list of unique target IDs.

The target IDs are sorted by the order that they first appear.

Returns (array): an array of integer target IDs.

update(*other*)
Overwrite or append new data.

Given another Spectra object, compare the fibermap information with the existing one. For spectra that already exist, overwrite existing data with the new values. For spectra that do not exist, append that data to the end of the spectral data.

Parameters **other** (Spectra) – the new data to add.

Returns nothing (object updated in place).

Note: if fibermap, scores and extra_catalog exist in the new data, they are appended to the existing tables. If those new tables have different columns, only columns with identical names will be appended. Spectra.meta is unchanged.

wavelength_grid(*band*)
Return the wavelength grid for a band.

Parameters **band** (*str*) – the name of the band.

Returns (array): an array containing the wavelength values.

desispec.spectra.**stack**(*speclist*)
Stack a list of spectra, return a new spectra object

Parameters **speclist** – list of Spectra objects

returns stacked Spectra object

Note: all input spectra must have the same bands, wavelength grid, and include or not the same optional elements (mask, fibermap, extra, ...). The returned Spectra have the meta from the first input Spectra.

Also see Spectra.update, which is less efficient but more flexible for handling heterogeneous inputs

1.8.57 desispec.trace_shifts

desispec.trace_shifts.**boxcar_extraction_from_filenames**(*image_filename*, *psf_filename*, *fibers=None*, *width=7*)
Fast boxcar extraction of spectra from a preprocessed image and a trace set

Parameters

- **image_filename** – input preprocessed fits filename
- **psf_filename** – input PSF fits filename

Optional: fibers : 1D np.array of int (default is all fibers, the first fiber is always = 0) width : extraction boxcar width, default is 7

Returns 2D np.array of shape (nfibers,n0=image.shape[0]), sum of pixel values per row of length=width per fiber ivar : 2D np.array of shape (nfibers,n0), ivar[f,j] = 1/(sum_[j,b:e] (1/image.ivar)), ivar=0 if at least 1 pixel in the row has image.ivar=0 or image.mask!=0 wave : 2D np.array of shape (nfibers,n0), determined from the traces

Return type flux

desispec.trace_shifts.**compute_dx_dy_using_psf**(*psf*, *image*, *fibers*, *lines*)

Computes trace shifts along x and y from a preprocessed image, a PSF (with trace coords), and a set of emission lines, by doing a forward model of the image. Calls compute_fiber_bundle_trace_shifts_using_psf.

Parameters

- **psf** – specter psf object
- **image** – DESI preprocessed image object
- **fibers** – 1D array with list of fibers
- **lines** – 1D array of wavelength of emission lines (in Angstrom)

Returns 1D array of x coordinates on CCD (axis=1 in numpy image array, AXIS=0 in FITS, cross-dispersion axis = fiber number direction) y : 1D array of y coordinates on CCD (axis=0 in numpy image array, AXIS=1 in FITS, wavelength dispersion axis) dx : 1D array of shifts along x coordinates on CCD dy : 1D array of shifts along y coordinates on CCD sx : 1D array of uncertainties on dx sy : 1D array of uncertainties on dy fiber : 1D array of fiber ID wave : 1D array of wavelength

Return type x

desispec.trace_shifts.**compute_dx_from_cross_dispersion_profiles**(*xcoef*, *ycoef*, *wavemin*, *wavemax*, *image*, *fibers*, *width=7*, *deg=2*, *image_rebin=4*)

Measure x offsets from a preprocessed image and a trace set

Parameters

- **xcoef** – 2D np.array of shape (nfibers,ncoef) containing Legendre coefficents for each fiber to convert wavelenght to XCCD
- **ycoef** – 2D np.array of shape (nfibers,ncoef) containing Legendre coefficents for each fiber to convert wavelenght to YCCD
- **wavemin** – float
- **wavemax** – float. wavemin and wavemax are used to define a reduced variable legx(wave,wavemin,wavemax)=2*(wave-wavemin)/(wavemax-wavemin)-1 used to compute the traces, xccd=legval(legx(wave,wavemin,wavemax),xtrace[fiber])
- **image** – DESI preprocessed image object

- **fibers** – 1D np.array of int (default is all fibers, the first fiber is always = 0)

Optional: width : extraction boxcar width, default is 5 deg : degree of polynomial fit as a function of y, only used to find and mask outliers image_rebin : rebinning of CCD rows to run faster (with rebin=4 loss of precision <0.01 pixel)

Returns 1D array of x coordinates on CCD (axis=1 in numpy image array, AXIS=0 in FITS, cross-dispersion axis = fiber number direction) y : 1D array of y coordinates on CCD (axis=0 in numpy image array, AXIS=1 in FITS, wavelength dispersion axis) dx : 1D array of shifts along x coordinates on CCD ex : 1D array of uncertainties on dx fiber : 1D array of fiber ID (first fiber = 0) wave : 1D array of wavelength

Return type x

desispec.trace_shifts.**compute_dy_from_spectral_cross_correlation**(*flux*, *wave*, *refflux*, *ivar=None*, *hw=3.0*, *calibrate=False*)

Measure y offsets from two spectra expected to be on the same wavelength grid. refflux is the assumed well calibrated spectrum. A relative flux calibration of the two spectra is done internally.

Parameters

- **flux** – 1D array of spectral flux as a function of wavelenght
- **wave** – 1D array of wavelength (in Angstrom)
- **refflux** – 1D array of reference spectral flux

Optional: ivar : 1D array of inverse variance of flux hw : half width in Angstrom of the cross-correlation chi2 scan, default=3A corresponding approximatly to 5 pixels for DESI

Returns 1D array of x coordinates on CCD (axis=1 in numpy image array, AXIS=0 in FITS, cross-dispersion axis = fiber number direction) y : 1D array of y coordinates on CCD (axis=0 in numpy image array, AXIS=1 in FITS, wavelength dispersion axis) dx : 1D array of shifts along x coordinates on CCD ex : 1D array of uncertainties on dx fiber : 1D array of fiber ID (first fiber = 0) wave : 1D array of wavelength

Return type x

desispec.trace_shifts.**compute_dy_from_spectral_cross_correlations_of_frame**(*flux*, *ivar*, *wave*, *xcoef*, *ycoef*, *wavemin*, *wavemax*, *reference_flux*, *n_wavelength_bins=4*)

Measures y offsets from a set of resampled spectra and a reference spectrum that are on the same wavelength grid. reference_flux is the assumed well calibrated spectrum. Calls compute_dy_from_spectral_cross_correlation per fiber

Parameters

- **flux** – 2D np.array of shape (nfibers,nwave)
- **ivar** – 2D np.array of shape (nfibers,nwave) , inverse variance of flux
- **wave** – 1D array of wavelength (in Angstrom) of size nwave
- **refflux** – 1D array of reference spectral flux of size nwave

Optional: n_wavelength_bins : number of bins along wavelength

Returns 1D array of x coordinates on CCD (axis=1 in numpy image array, AXIS=0 in FITS, cross-dispersion axis = fiber number direction) y : 1D array of y coordinates on CCD (axis=0 in numpy image array, AXIS=1 in FITS, wavelength dispersion axis) dy : 1D array of shifts along y coordinates on CCD ey : 1D array of uncertainties on dy fiber : 1D array of fiber ID (first fiber = 0) wave : 1D array of wavelength

Return type x

desispec.trace_shifts.**compute_dy_using_boxcar_extraction**(*xytraceset*, *image*, *fibers*, *width=7*, *degyy=2*)

Measures y offsets (internal wavelength calibration) from a preprocessed image and a trace set using a cross-correlation of boxcar extracted spectra. Uses boxcar_extraction , resample_boxcar_frame , compute_dy_from_spectral_cross_correlations_of_frame

Parameters

- **xytraceset** – XYTraceset object
- **image** – DESI preprocessed image object

Optional: fibers : 1D np.array of int (default is all fibers, the first fiber is always = 0) width : int, extraction boxcar width, default is 7 degyy : int, degree of polynomial fit of shifts as a function of y, used to reject outliers.

Returns 1D array of x coordinates on CCD (axis=1 in numpy image array, AXIS=0 in FITS, cross-dispersion axis = fiber number direction) y : 1D array of y coordinates on CCD (axis=0 in numpy image array, AXIS=1 in FITS, wavelength dispersion axis) dy : 1D array of shifts along y coordinates on CCD ey : 1D array of uncertainties on dy fiber : 1D array of fiber ID (first fiber = 0) wave : 1D array of wavelength

Return type x

desispec.trace_shifts.**compute_fiber_bundle_trace_shifts_using_psf**(*fibers*, *line*, *psf*, *image*, *maxshift=2.0*)

Computes trace shifts along x and y from a preprocessed image, a PSF (with trace coords), and a given emission line, by doing a forward model of the image.

Parameters

- **fibers** – 1D array with list of fibers
- **line** – float, wavelength of an emission line (in Angstrom)
- **psf** – specter psf object
- **image** – DESI preprocessed image object

Optional: maxshift : float maximum shift in pixels for 2D chi2 scan

Returns 1D array of x coordinates on CCD (axis=1 in numpy image array, AXIS=0 in FITS, cross-dispersion axis = fiber number direction) y : 1D array of y coordinates on CCD (axis=0 in numpy image array, AXIS=1 in FITS, wavelength dispersion axis) dx : 1D array of shifts along x coordinates on CCD dy : 1D array of shifts along y coordinates on CCD sx : 1D array of uncertainties on dx sy : 1D array of uncertainties on dy

Return type x

desispec.trace_shifts.**legx**(*wave*, *wavemin*, *wavemax*)
Reduced coordinate (range [-1,1]) for calls to legval and legfit

Parameters

- **wave** – ND np.array
- **wavemin** – float, min. val
- **wavemax** – float, max. val

Returns array of same shape as wave

desispec.trace_shifts.**monomials**(*x*, *y*, *degx*, *degy*)
Computes monomials as a function of x and y of a 2D polynomial of degrees degx and degy

Parameters

- **x** – ND array
- **y** – ND array of same shape as x
- **degx** – int (>=0), polynomial degree along x
- **degy** – int (>=0), polynomial degree along y

Returns : monomials : ND array of shape ((degx+1)*(degy+1) , x shape)

desispec.trace_shifts.**polynomial_fit**(*z*, *ez*, *xx*, *yy*, *degx*, *degy*)
Computes and 2D polynomial fit of z as a function of (x,y) of degrees degx and degy

Parameters

- **z** – ND array
- **ez** – ND array of same shape as z, uncertainties on z
- **x** – ND array of same shape as z
- **y** – ND array of same shape as z
- **degx** – int (>=0), polynomial degree along x
- **degy** – int (>=0), polynomial degree along y

Returns 1D array of size (degx+1)*(degy+1) with polynomial coefficients (as defined by routine monomials) covariance : 2D array of covariance of coeff error_floor : float , extra uncertainty needed to get chi2/ndf=1 polval : ND array of same shape as z with values of pol(x,y) mask : ND array of same shape as z indicating the masked data points in the fit

Return type coeff

desispec.trace_shifts.**recompute_legendre_coefficients**(*xcoef*, *ycoef*, *wavemin*, *wavemax*, *degxx*, *degxy*, *degyx*, *degyy*, *dx_coeff*, *dy_coeff*)
Modifies legendre coefficients of an input trace set using polynomial coefficents (as defined by the routine monomials)

Parameters

- **xcoef** – 2D np.array of shape (nfibers,ncoef) containing Legendre coefficents for each fiber to convert wavelenght to XCCD
- **ycoef** – 2D np.array of shape (nfibers,ncoef) containing Legendre coefficents for each fiber to convert wavelenght to YCCD
- **wavemin** – float
- **wavemax** – float. wavemin and wavemax are used to define a reduced variable legx(wave,wavemin,wavemax)=2*(wave-wavemin)/(wavemax-wavemin)-1 used to compute the traces, xccd=legval(legx(wave,wavemin,wavemax),xtrace[fiber])
- **degxx** – int, degree of polynomial for x shifts as a function of x (x is axis=1 in numpy image array, AXIS=0 in FITS, cross-dispersion axis = fiber number direction)
- **degxy** – int, degree of polynomial for x shifts as a function of y (y is axis=0 in numpy image array, AXIS=1 in FITS, wavelength dispersion axis)
- **degyx** – int, degree of polynomial for y shifts as a function of x
- **degyy** – int, degree of polynomial for y shifts as a function of y
- **dx_coeff** – 1D np.array of polynomial coefficients of size (degxx*degxy) as defined by the routine monomials.
- **dy_coeff** – 1D np.array of polynomial coefficients of size (degyx*degyy) as defined by the routine monomials.

Returns 2D np.array of shape (nfibers,ncoef) with modified Legendre coefficents ycoef : 2D np.array of shape (nfibers,ncoef) with modified Legendre coefficents

Return type xcoef

desispec.trace_shifts.**resample_boxcar_frame**(*frame_flux*, *frame_ivar*, *frame_wave*, *oversampling=2*)

Resamples the spectra in a frame obtained with boxcar extraction to the same wavelength grid, with oversampling. Uses resample_flux routine.

Parameters

- **frame_flux** – 2D np.array of shape (nfibers,nwave), sum of pixel values per row of length=width per fiber
- **frame_ivar** – 2D np.array of shape (nfibers,nwave), ivar[f,j] = 1/(sum_[j,b:e] (1/image.ivar)), ivar=0 if at least 1 pixel in the row has image.ivar=0 or image.mask!=0
- **frame_wave** – 2D np.array of shape (nfibers,nwave), determined from the traces

Optional: oversampling : int , oversampling factor , default is 2

Returns 2D np.array of shape (nfibers,nwave*oversampling) ivar : 2D np.array of shape (nfibers,nwave*oversampling) frame_wave : 1D np.array of size (nwave*oversampling)

Return type flux

desispec.trace_shifts.**shift_ycoef_using_external_spectrum**(*psf*, *xytraceset*, *image*, *fibers*, *spectrum_filename*, *degyy=2*, *width=7*)

Measure y offsets (external wavelength calibration) from a preprocessed image , a PSF + trace set using a cross-correlation of boxcar extracted spectra and an external well-calibrated spectrum. The PSF shape is used to

convolve the input spectrum. It could also be used to correct for the PSF asymetry (disabled for now). A relative flux calibration of the spectra is performed internally.

Parameters

- **psf** – specter PSF
- **xytraceset** – XYTraceset object
- **image** – DESI preprocessed image object
- **fibers** – 1D np.array of fiber indices
- **spectrum_filename** – path to input spectral file (read with np.loadtxt , first column is wavelength (in vacuum and Angstrom) , second column in flux (arb. units)

Optional: width : int, extraction boxcar width, default is 7 degyy : int, degree of polynomial fit of shifts as a function of y, used to reject outliers.

Returns 2D np.array of same shape as input, with modified Legendre coefficents for each fiber to convert wavelenght to YCCD

Return type ycoef

desispec.trace_shifts.**write_traces_in_psf**(*input_psf_filename*, *output_psf_filename*, *xytraceset*)

Writes traces in a PSF.

Parameters

- **input_psf_filename** – Path to input fits file which has to contain XTRACE and YTRACE HDUs
- **output_psf_filename** – Path to output fits file which has to contain XTRACE and YTRACE HDUs
- **xytraceset** – xytraceset

Utility functions for desispec

desispec.util.**combine_ivar**(*ivar1*, *ivar2*)

Returns the combined inverse variance of two inputs, making sure not to divide by 0 in the process.

ivar1 and ivar2 may be scalar or ndarray but must have the same dimensions

desispec.util.**dateobs2night**(*dateobs*)

Convert DATE-OBS ISO8601 UTC string to YEARMMDD int night of KPNO sunset

desispec.util.**header2night**(*header*)

Return YEARMMDD night from FITS header, handling common problems

desispec.util.**healpix_degrade_fixed**(*nside*, *pixel*)

Degrade a NEST ordered healpix pixel with a fixed ratio.

This degrades the pixel to a lower nside value that is fixed to half the healpix "factor".

Parameters

- **nside** (*int*) – a valid NSIDE value.
- **pixel** (*int*) – the NESTED pixel index.

Returns (tuple): a tuple of ints, where the first value is the new NSIDE and the second value is the degraded pixel index.

desispec.util.**itemindices**(*a*)
Return dict[key] -> list of indices i where a[i] == key

Parameters **a** – array-like of hashable values

Return dict[key] -> list of indices i where a[i] == key

The dict keys are inserted in the order that they first appear in a, and the value lists of indices are sorted

e.g. itemindices([10,30,20,30]) -> {10: [0], 30: [1, 3], 20: [2]}

desispec.util.**mask32**(*mask*)
Return an input mask as unsigned 32-bit

Raises ValueError if 64-bit input can't be cast to 32-bit without losing info (i.e. if it contains values > 2**32-1)

desispec.util.**mjd2night**(*mjd*)
Convert MJD to YEARMMDD int night of KPNO sunset

desispec.util.**mpi_count_failures**(*num_cmd*, *num_err*, *comm=None*)
Sum num_cmd and num_err across MPI ranks

Parameters

- **num_cmd** (*int*) – number of commands run
- **num_err** (*int*) – number of failures

Options: comm: mpi4py communicator

Returns sum(num_cmd), sum(num_err) summed across all MPI ranks

If `comm` is None, returns input num_cmd, num_err

desispec.util.**night2ymd**(*night*)
parse night YEARMMDD string into tuple of integers (year, month, day)

desispec.util.**option_list**(*opts*)
Convert key, value pairs into command-line options.

Parameters **opts** (*dict-like*) – Convert a dictionary into command-line options.

Returns A list of command-line options.

Return type `list`

desispec.util.**ordered_unique**(*ar*, *return_index=False*)
Find the unique elements of an array in the order they first appear

Like numpy.unique, but preserves original order instead of sorting

Parameters **ar** – array-like data to find unique elements

Options: return_index: if True also return indices in ar where items first appear

desispec.util.**parse_fibers**(*fiber_string*, *include_end=False*)
Short func that parses a string containing a comma separated list of integers, which can include ":" or ".." or "-" labeled ranges

Parameters **fiber_string** (*str*) – list of integers or integer ranges

Options: include_end (bool): if True, include end-value in ranges

Returns (array 1-D): 1D numpy array listing all of the integers given in the list, including enumerations of ranges given.

Note: this follows python-style ranges, i,e, 1:5 or 1..5 returns 1, 2, 3, 4 unless *include_end* is True, which then returns 1,2,3,4,5

desispec.util.**parse_int_args**(*arg_string*, *include_end=False*)
Short func that parses a string containing a comma separated list of integers, which can include ":" or ".." or "-" labeled ranges

Parameters **arg_string** (*str*) – list of integers or integer ranges

Options: include_end (bool): if True, include end-value in ranges

Returns (array 1-D): 1D numpy array listing all of the integers given in the list, including enumerations of ranges given.

Note: this follows python-style ranges, i,e, 1:5 or 1..5 returns 1,2,3,4 unless *include_end* is True, which then returns 1,2,3,4,5

desispec.util.**pid_exists**(*pid*)
Check whether pid exists in the current process table.

UNIX only. Should work the same as psutil.pid_exists().

Parameters **pid** (*int*) – A process ID.

Returns `True` if the process exists in the current process table.

Return type pid_exists (bool)

desispec.util.**runcmd**(*cmd*, *args=None*, *inputs=[]*, *outputs=[]*, *clobber=False*)
Runs a command, checking for inputs and outputs

Parameters

- **cmd** – command string to run with subprocess.call()
- **inputs** – list of filename inputs that must exist before running
- **outputs** – list of output filenames that should be created
- **clobber** – if True, run even if outputs already exist

Returns error code from command or input/output checking; 0 is good

Notes

If any inputs are missing, don't run cmd. If outputs exist and have timestamps after all inputs, don't run cmd.

desispec.util.**sprun**(*com*, *capture=False*, *input=None*)
Run a command with subprocess and handle errors.

This runs a command and returns the lines of STDOUT as a list. Any contents of STDERR are logged. If an OSError is raised by the child process, that is also logged. If another exception is raised by the child process, the traceback from the child process is printed.

Parameters

- **com** (*list*) – the command to run.
- **capture** (*bool*) – if True, return the stdout contents.

- **input** (*str*) – the string data (can include embedded newlines) to write to the STDIN of the child process.

Returns

the return code and optionally the lines of STDOUT from the child process.

Return type tuple(int, (list))

desispec.util.**ymd2night**(*year*, *month*, *day*)
convert year, month, day integers into cannonical YEARMMDD night string

1.8.58 desispec.workflow.schedule

Tools for scheduling MPI jobs using mpi4py

1.8.59 desispec.xytraceset

Lightweight wrapper class for trace coordinates and wavelength solution, to be returned by *read_xytraceset()*.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

d

desispec, 49
desispec.averagefluxcalibration, 49
desispec.bootcalib, 49
desispec.calibfinder, 55
desispec.coaddition, 56
desispec.cosmics, 58
desispec.database, 59
desispec.database.metadata, 60
desispec.database.redshift, 62
desispec.database.util, 64
desispec.fiberflat, 65
desispec.fluxcalibration, 66
desispec.frame, 71
desispec.image, 71
desispec.interpolation, 71
desispec.io, 72
desispec.io.download, 73
desispec.io.fiberflat, 73
desispec.io.fibermap, 74
desispec.io.filters, 75
desispec.io.fluxcalibration, 76
desispec.io.frame, 77
desispec.io.image, 78
desispec.io.meta, 78
desispec.io.params, 81
desispec.io.qa, 81
desispec.io.raw, 83
desispec.io.sky, 84
desispec.io.spectra, 84
desispec.io.util, 86
desispec.io.xytraceset, 91
desispec.linalg, 91
desispec.log, 92
desispec.maskbits, 92
desispec.parallel, 93
desispec.pipeline, 95
desispec.pipeline.control, 95
desispec.pipeline.db, 101
desispec.pipeline.defs, 105
desispec.pipeline.prod, 106
desispec.pipeline.run, 107
desispec.pipeline.scriptgen, 109
desispec.pipeline.tasks, 111
desispec.pipeline.tasks.base, 111
desispec.pipeline.tasks.cframe, 115
desispec.pipeline.tasks.extract, 116
desispec.pipeline.tasks.fiberflat, 116
desispec.pipeline.tasks.fiberflatnight, 116
desispec.pipeline.tasks.fibermap, 117
desispec.pipeline.tasks.fluxcalib, 117
desispec.pipeline.tasks.preproc, 117
desispec.pipeline.tasks.psf, 118
desispec.pipeline.tasks.psfnight, 118
desispec.pipeline.tasks.qadata, 119
desispec.pipeline.tasks.rawdata, 119
desispec.pipeline.tasks.redshift, 120
desispec.pipeline.tasks.sky, 120
desispec.pipeline.tasks.spectra, 121
desispec.pipeline.tasks.starfit, 121
desispec.pipeline.tasks.traceshift, 122
desispec.pixgroup, 122
desispec.preproc, 124
desispec.qa, 128
desispec.qa.html, 128
desispec.qa.qa_brick, 129
desispec.qa.qa_exposure, 129
desispec.qa.qa_frame, 129
desispec.qa.qa_multiexp, 130
desispec.qa.qa_night, 130
desispec.qa.qa_plots, 130
desispec.qa.qa_plots_ql, 133
desispec.qa.qa_prod, 135
desispec.qa.qa_quicklook, 135
desispec.qa.qalib, 135
desispec.qa.utils, 138
desispec.qproc, 139
desispec.qproc.io, 139

desispec.qproc.qextract, 139
desispec.qproc.qfiberflat, 140
desispec.qproc.qframe, 140
desispec.qproc.qsky, 140
desispec.quicklook, 140
desispec.quicklook.arcprocess, 140
desispec.quicklook.merger, 140
desispec.quicklook.palib, 140
desispec.quicklook.pas, 141
desispec.quicklook.procalgs, 141
desispec.quicklook.qas, 142
desispec.quicklook.ql_plotlib, 144
desispec.quicklook.qlboxcar, 143
desispec.quicklook.qlconfig, 143
desispec.quicklook.qlexceptions, 144
desispec.quicklook.qlheartbeat, 144
desispec.quicklook.qllogger, 144
desispec.quicklook.qlpsf, 145
desispec.quicklook.qlresolution, 145
desispec.quicklook.quickfiberflat, 146
desispec.quicklook.quicklook, 146
desispec.quicklook.quicksky, 146
desispec.resolution, 147
desispec.scripts, 148
desispec.scripts.average_fiberflat, 148
desispec.scripts.bootcalib, 148
desispec.scripts.extract, 148
desispec.scripts.fiberflat, 148
desispec.scripts.fluxcalibration, 148
desispec.scripts.group_spectra, 148
desispec.scripts.mergebundles, 148
desispec.scripts.night, 148
desispec.scripts.pipe, 148
desispec.scripts.pipe_exec, 148
desispec.scripts.preproc, 148
desispec.scripts.procexp, 149
desispec.scripts.qa_exposure, 149
desispec.scripts.qa_frame, 149
desispec.scripts.qa_prod, 149
desispec.scripts.quicklook, 149
desispec.scripts.rejectcosmics, 150
desispec.scripts.sky, 150
desispec.scripts.skysubresid, 150
desispec.scripts.specex, 150
desispec.scripts.specscore, 150
desispec.scripts.stdstars, 150
desispec.scripts.trace_shifts, 151
desispec.scripts.update_spectra, 151
desispec.sky, 151
desispec.specscore, 153
desispec.spectra, 154
desispec.trace_shifts, 156
desispec.util, 162
desispec.workflow.schedule, 165
desispec.xytraceset, 165

Index

Symbols

_auth() (*in module desispec.io.download*), 73
_background() (*in module desispec.preproc*), 124
_clipped_std_bias() (*in module desispec.preproc*), 124
_coarse_overlapping_bricks() (*desispec.database.metadata.Tile method*), 60
_compute_coef() (*in module desispec.fluxcalibration*), 66
_constants() (*desispec.database.metadata.Tile method*), 60
_create() (*desispec.pipeline.tasks.base.BaseTask method*), 111
_deps() (*desispec.pipeline.tasks.cframe.TaskCFrame method*), 115
_deps() (*desispec.pipeline.tasks.extract.TaskExtract method*), 116
_deps() (*desispec.pipeline.tasks.fiberflat.TaskFiberflat method*), 116
_deps() (*desispec.pipeline.tasks.fiberflatnight.TaskFiberflatNight method*), 116
_deps() (*desispec.pipeline.tasks.fibermap.TaskFibermap method*), 117
_deps() (*desispec.pipeline.tasks.fluxcalib.TaskFluxCalib method*), 117
_deps() (*desispec.pipeline.tasks.preproc.TaskPreproc method*), 118
_deps() (*desispec.pipeline.tasks.psf.TaskPSF method*), 118
_deps() (*desispec.pipeline.tasks.psfnight.TaskPSFNight method*), 118
_deps() (*desispec.pipeline.tasks.qadata.TaskQAData method*), 119
_deps() (*desispec.pipeline.tasks.rawdata.TaskRawdata method*), 119
_deps() (*desispec.pipeline.tasks.redshift.TaskRedshift method*), 120
_deps() (*desispec.pipeline.tasks.sky.TaskSky method*), 120
_deps() (*desispec.pipeline.tasks.spectra.TaskSpectra method*), 121
_deps() (*desispec.pipeline.tasks.starfit.TaskStarFit method*), 121
_deps() (*desispec.pipeline.tasks.traceshift.TaskTraceShift method*), 122
_dict2ndarray() (*in module desispec.io.util*), 86
_extract_and_save() (*in module desispec.scripts.extract*), 148
_func() (*in module desispec.fluxcalibration*), 67
_func2() (*in module desispec.fluxcalibration*), 67
_gauss_pix() (*in module desispec.resolution*), 147
_get_mags() (*in module desispec.qa.qalib*), 136
_get_slice() (*desispec.spectra.Spectra method*), 155
_global_background() (*in module desispec.preproc*), 124
_insert() (*desispec.pipeline.tasks.base.BaseTask method*), 111
_load_smsp() (*in module desispec.calibfinder*), 55
_map_download() (*in module desispec.io.download*), 73
_model_variance() (*in module desispec.sky*), 151
_option_dict() (*desispec.pipeline.tasks.psfnight.TaskPSFNight method*), 118
_option_list() (*desispec.pipeline.tasks.cframe.TaskCFrame method*), 115
_option_list() (*desispec.pipeline.tasks.extract.TaskExtract method*), 116
_option_list() (*desispec.pipeline.tasks.fiberflat.TaskFiberflat method*), 116
_option_list() (*desispec.pipeline.tasks.fiberflatnight.TaskFiberflatNight method*), 116
_option_list() (*desispec.pipeline.tasks.fluxcalib.TaskFluxCalib*

method), 117
_option_list() (*desispec.pipeline.tasks.preproc.TaskPreproc method*), 118
_option_list() (*desispec.pipeline.tasks.psf.TaskPSF method*), 118
_option_list() (*desispec.pipeline.tasks.psfnight.TaskPSFNight method*), 118
_option_list() (*desispec.pipeline.tasks.qadata.TaskQAData method*), 119
_option_list() (*desispec.pipeline.tasks.redshift.TaskRedshift method*), 120
_option_list() (*desispec.pipeline.tasks.sky.TaskSky method*), 120
_option_list() (*desispec.pipeline.tasks.starfit.TaskStarFit method*), 121
_option_list() (*desispec.pipeline.tasks.traceshift.TaskTraceShift method*), 122
_overscan() (*in module desispec.preproc*), 124
_paths() (*desispec.pipeline.tasks.cframe.TaskCFrame method*), 115
_paths() (*desispec.pipeline.tasks.extract.TaskExtract method*), 116
_paths() (*desispec.pipeline.tasks.fiberflat.TaskFiberflat method*), 116
_paths() (*desispec.pipeline.tasks.fiberflatnight.TaskFiberflatNight method*), 116
_paths() (*desispec.pipeline.tasks.fibermap.TaskFibermap method*), 117
_paths() (*desispec.pipeline.tasks.fluxcalib.TaskFluxCalib method*), 117
_paths() (*desispec.pipeline.tasks.preproc.TaskPreproc method*), 118
_paths() (*desispec.pipeline.tasks.psf.TaskPSF method*), 118
_paths() (*desispec.pipeline.tasks.psfnight.TaskPSFNight method*), 119
_paths() (*desispec.pipeline.tasks.qadata.TaskQAData method*), 119
_paths() (*desispec.pipeline.tasks.rawdata.TaskRawdata method*), 119
_paths() (*desispec.pipeline.tasks.redshift.TaskRedshift method*), 120
_paths() (*desispec.pipeline.tasks.sky.TaskSky method*), 120
_paths() (*desispec.pipeline.tasks.spectra.TaskSpectra method*), 121
_paths() (*desispec.pipeline.tasks.starfit.TaskStarFit method*), 121
_paths() (*desispec.pipeline.tasks.traceshift.TaskTraceShift method*), 122
_preproc_file_kwargs_wrapper() (*in module desispec.scripts.preproc*), 149
_reject_cosmic_rays_ala_sdss_single() (*in module desispec.cosmics*), 58
_reject_cosmic_rays_ala_sdss_single_numba() (*in module desispec.cosmics*), 58
_retrieve() (*desispec.pipeline.tasks.base.BaseTask method*), 111
_run() (*desispec.pipeline.tasks.cframe.TaskCFrame method*), 115
_run() (*desispec.pipeline.tasks.extract.TaskExtract method*), 116
_run() (*desispec.pipeline.tasks.fiberflat.TaskFiberflat method*), 116
_run() (*desispec.pipeline.tasks.fiberflatnight.TaskFiberflatNight method*), 117
_run() (*desispec.pipeline.tasks.fibermap.TaskFibermap method*), 117
_run() (*desispec.pipeline.tasks.fluxcalib.TaskFluxCalib method*), 117
_run() (*desispec.pipeline.tasks.preproc.TaskPreproc method*), 118
_run() (*desispec.pipeline.tasks.psf.TaskPSF method*), 118
_run() (*desispec.pipeline.tasks.psfnight.TaskPSFNight method*), 119
_run() (*desispec.pipeline.tasks.qadata.TaskQAData method*), 119
_run() (*desispec.pipeline.tasks.rawdata.TaskRawdata method*), 119
_run() (*desispec.pipeline.tasks.redshift.TaskRedshift method*), 120
_run() (*desispec.pipeline.tasks.sky.TaskSky method*), 120
_run() (*desispec.pipeline.tasks.spectra.TaskSpectra method*), 121
_run() (*desispec.pipeline.tasks.starfit.TaskStarFit method*), 121
_run() (*desispec.pipeline.tasks.traceshift.TaskTraceShift method*), 122
_run_cli() (*desispec.pipeline.tasks.cframe.TaskCFrame method*), 115
_run_cli() (*desispec.pipeline.tasks.extract.TaskExtract method*), 116
_run_cli() (*desispec.pipeline.tasks.fiberflat.TaskFiberflat method*), 116
_run_cli() (*desispec.pipeline.tasks.fiberflatnight.TaskFiberflatNight method*), 117
_run_cli() (*desispec.pipeline.tasks.fibermap.TaskFibermap method*), 117
_run_cli() (*desispec.pipeline.tasks.fluxcalib.TaskFluxCalib*

method), 117
_run_cli() (*desispec.pipeline.tasks.preproc.TaskPreproc method*), 118
_run_cli() (*desispec.pipeline.tasks.psf.TaskPSF method*), 118
_run_cli() (*desispec.pipeline.tasks.psfnight.TaskPSFNight method*), 119
_run_cli() (*desispec.pipeline.tasks.qadata.TaskQAData method*), 119
_run_cli() (*desispec.pipeline.tasks.rawdata.TaskRawdata method*), 119
_run_cli() (*desispec.pipeline.tasks.redshift.TaskRedshift method*), 120
_run_cli() (*desispec.pipeline.tasks.sky.TaskSky method*), 121
_run_cli() (*desispec.pipeline.tasks.spectra.TaskSpectra method*), 121
_run_cli() (*desispec.pipeline.tasks.starfit.TaskStarFit method*), 122
_run_cli() (*desispec.pipeline.tasks.traceshift.TaskTraceShift method*), 122
_run_defaults() (*desispec.pipeline.tasks.cframe.TaskCFrame method*), 115
_run_defaults() (*desispec.pipeline.tasks.extract.TaskExtract method*), 116
_run_defaults() (*desispec.pipeline.tasks.fiberflat.TaskFiberflat method*), 116
_run_defaults() (*desispec.pipeline.tasks.fiberflatnight.TaskFiberflatNight method*), 117
_run_defaults() (*desispec.pipeline.tasks.fibermap.TaskFibermap method*), 117
_run_defaults() (*desispec.pipeline.tasks.fluxcalib.TaskFluxCalib method*), 117
_run_defaults() (*desispec.pipeline.tasks.preproc.TaskPreproc method*), 118
_run_defaults() (*desispec.pipeline.tasks.psf.TaskPSF method*), 118
_run_defaults() (*desispec.pipeline.tasks.psfnight.TaskPSFNight method*), 119
_run_defaults() (*desispec.pipeline.tasks.qadata.TaskQAData method*), 119
_run_defaults() (*desispec.pipeline.tasks.rawdata.TaskRawdata method*), 120
_run_defaults() (*desispec.pipeline.tasks.redshift.TaskRedshift method*), 120
_run_defaults() (*desispec.pipeline.tasks.sky.TaskSky method*), 121
_run_defaults() (*desispec.pipeline.tasks.spectra.TaskSpectra method*), 121
_run_defaults() (*desispec.pipeline.tasks.starfit.TaskStarFit method*), 122
_run_defaults() (*desispec.pipeline.tasks.traceshift.TaskTraceShift method*), 122
_run_max_mem_proc() (*desispec.pipeline.tasks.base.BaseTask method*), 111
_run_max_mem_task() (*desispec.pipeline.tasks.base.BaseTask method*), 111
_run_max_procs() (*desispec.pipeline.tasks.cframe.TaskCFrame method*), 115
_run_time() (*desispec.pipeline.tasks.cframe.TaskCFrame method*), 115
_savgol_clipped() (*in module desispec.preproc*), 124
_set_fibermap_columns() (*in module desispec.io.fibermap*), 74
_smooth_template() (*in module desispec.fluxcalibration*), 67
_sort_and_symmeterize() (*in module desispec.resolution*), 147
_state_get() (*desispec.pipeline.tasks.base.BaseTask method*), 111
_state_set() (*desispec.pipeline.tasks.base.BaseTask method*), 111
_supports_memmap() (*in module desispec.io.util*), 86
_unweighted_resample() (*in module desispec.interpolation*), 71

A

add_columns() (*in module desispec.io.util*), 86
add_missing_frames() (*in module desispec.pixgroup*), 123
addkeys() (*in module desispec.io.util*), 86
all_task_types() (*in module desispec.pipeline.db*), 104
all_tasks() (*in module desispec.pipeline.db*), 104
ampregion() (*in module desispec.qa.qalib*), 136
angstroms_per_pixel() (*desispec.quicklook.qlpsf.PSF method*), 145

apply_fiberflat() (*in module desispec.fiberflat*), 65
apply_fiberflat() (*in module desispec.quicklook.quickfiberflat*), 146
apply_flux_calibration() (*in module desispec.fluxcalibration*), 67
apply_flux_calibration() (*in module desispec.quicklook.palib*), 140
ApplyFiberFlat (*class in desispec.quicklook.procalgs*), 141
ApplyFiberFlat_QL (*class in desispec.quicklook.procalgs*), 141
ApplyFiberFlat_QP (*class in desispec.quicklook.procalgs*), 141
ApplyFluxCalibration (*class in desispec.quicklook.procalgs*), 141
applySmoothingFilter() (*in module desispec.fluxcalibration*), 67
area (*desispec.database.metadata.Tile attribute*), 60
assemble_fibermap() (*in module desispec.io.fibermap*), 74
autocalib_fiberflat() (*in module desispec.fiberflat*), 65
average_fiberflat() (*in module desispec.fiberflat*), 65

B

badfibers() (*in module desispec.calibfinder*), 55
bands (*desispec.spectra.Spectra attribute*), 155
barycentric_correction_multiplicative_factor() (*in module desispec.scripts.extract*), 148
BaseTask (*class in desispec.pipeline.tasks.base*), 111
batch_nersc() (*in module desispec.pipeline.scriptgen*), 110
batch_shell() (*in module desispec.pipeline.scriptgen*), 110
Bias_From_Overscan (*class in desispec.qa.qa_quicklook*), 135
bootcalib() (*in module desispec.bootcalib*), 49
boxcar_extraction_from_filenames() (*in module desispec.trace_shifts*), 156
BoxcarExtract (*class in desispec.quicklook.procalgs*), 141
Brick (*class in desispec.database.metadata*), 60
brick_offset() (*desispec.database.metadata.Tile method*), 60
brick_redrock() (*in module desispec.qa.qa_plots*), 130
BrickStatus (*class in desispec.database.metadata*), 60

C

calc_overscan() (*in module desispec.preproc*), 125
Calc_XWSigma (*class in desispec.qa.qa_quicklook*), 135
Calculate_SNR (*class in desispec.qa.qa_quicklook*), 135
calculate_throughput_corrections() (*in module desispec.sky*), 151
calib() (*in module desispec.qa.html*), 128
calib_exp() (*in module desispec.qa.html*), 128
camword_to_spectros() (*in module desispec.io.util*), 87
camword_union() (*in module desispec.io.util*), 87
chain() (*in module desispec.pipeline.control*), 95
check_config() (*in module desispec.quicklook.qlconfig*), 144
Check_FiberFlat (*class in desispec.qa.qa_quicklook*), 135
Check_HDUs (*class in desispec.qa.qa_quicklook*), 135
Check_Resolution (*class in desispec.qa.qa_quicklook*), 135
check_tasks() (*in module desispec.pipeline.control*), 96
check_tasks() (*in module desispec.pipeline.db*), 104
checkgzip() (*in module desispec.io.util*), 87
cholesky_invert() (*in module desispec.linalg*), 91
cholesky_solve() (*in module desispec.linalg*), 91
cholesky_solve_and_invert() (*in module desispec.linalg*), 91
circum_square (*desispec.database.metadata.Tile attribute*), 60
cleanup() (*desispec.pipeline.db.DataBase method*), 101
cleanup() (*in module desispec.pipeline.control*), 96
coadd() (*in module desispec.coaddition*), 56
coadd_cameras() (*in module desispec.coaddition*), 56
coadd_fibermap() (*in module desispec.coaddition*), 56
combine_ivar() (*in module desispec.util*), 162
compare_fiberassign() (*in module desispec.io.fibermap*), 74
compatible() (*in module desispec.scripts.specex*), 150
compute_background_between_fiber_blocks() (*in module desispec.preproc*), 125
compute_coadd_scores() (*in module desispec.specscore*), 153
compute_coadd_tsnr_scores() (*in module desispec.specscore*), 154
compute_dx_dy_using_psf() (*in module desispec.trace_shifts*), 157
compute_dx_from_cross_dispersion_profiles() (*in module desispec.trace_shifts*), 157
compute_dy_from_spectral_cross_correlation() (*in module desispec.trace_shifts*), 158

compute_dy_from_spectral_cross_correlations_of_frame() (*in module desispec.trace_shifts*), 158
compute_dy_using_boxcar_extraction() (*in module desispec.trace_shifts*), 159
compute_fiber_bundle_trace_shifts_using_psf() (*in module desispec.trace_shifts*), 159
compute_fiberflat() (*in module desispec.fiberflat*), 65
compute_fiberflat() (*in module desispec.quicklook.quickfiberflat*), 146
compute_flux_calibration() (*in module desispec.fluxcalibration*), 67
compute_frame_scores() (*in module desispec.specscore*), 154
compute_non_uniform_sky() (*in module desispec.sky*), 151
compute_overscan_step() (*in module desispec.preproc*), 125
compute_polynomial_times_sky() (*in module desispec.sky*), 151
compute_sky() (*in module desispec.quicklook.quicksky*), 146
compute_sky() (*in module desispec.sky*), 152
compute_uniform_sky() (*in module desispec.sky*), 152
ComputeFiberflat (*class in desispec.quicklook.procalgs*), 142
ComputeFiberflat_QL (*class in desispec.quicklook.procalgs*), 142
ComputeFiberflat_QP (*class in desispec.quicklook.procalgs*), 142
ComputeSky (*class in desispec.quicklook.procalgs*), 142
ComputeSky_QL (*class in desispec.quicklook.procalgs*), 142
Config (*class in desispec.quicklook.qlconfig*), 143
continuum() (*in module desispec.qa.qalib*), 136
convert_dateobs() (*in module desispec.database.util*), 64
cos_radius (*desispec.database.metadata.Tile attribute*), 60
Count_Pixels (*class in desispec.qa.qa_quicklook*), 135
count_task_states() (*desispec.pipeline.db.DataBase method*), 102
countbins() (*in module desispec.qa.qalib*), 136
countpix() (*in module desispec.qa.qalib*), 136
CountSpectralBins (*class in desispec.qa.qa_quicklook*), 135
create() (*desispec.pipeline.tasks.base.BaseTask method*), 111
create() (*in module desispec.pipeline.control*), 96
create_camword() (*in module desispec.io.util*), 87

D

DataBase (*class in desispec.pipeline.db*), 101
DataBasePostgres (*class in desispec.pipeline.db*), 103
DataBaseSqlite (*class in desispec.pipeline.db*), 104
dateobs2night() (*in module desispec.util*), 162
decode_camword() (*in module desispec.io.util*), 87
decorrelate_divide_and_conquer() (*in module desispec.coaddition*), 57
default_nproc (*in module desispec.parallel*), 93
deps() (*desispec.pipeline.tasks.base.BaseTask method*), 111
DESI_SPECTRO_CALIB, 38
desispec (*module*), 49
desispec.averagefluxcalibration (*module*), 49
desispec.bootcalib (*module*), 49
desispec.calibfinder (*module*), 55
desispec.coaddition (*module*), 56
desispec.cosmics (*module*), 58
desispec.database (*module*), 59
desispec.database.metadata (*module*), 60
desispec.database.redshift (*module*), 62
desispec.database.util (*module*), 64
desispec.fiberflat (*module*), 65
desispec.fluxcalibration (*module*), 66
desispec.frame (*module*), 71
desispec.image (*module*), 71
desispec.interpolation (*module*), 71
desispec.io (*module*), 72
desispec.io.download (*module*), 73
desispec.io.fiberflat (*module*), 73
desispec.io.fibermap (*module*), 74
desispec.io.filters (*module*), 75
desispec.io.fluxcalibration (*module*), 76
desispec.io.frame (*module*), 77
desispec.io.image (*module*), 78
desispec.io.meta (*module*), 78
desispec.io.params (*module*), 81
desispec.io.qa (*module*), 81
desispec.io.raw (*module*), 83
desispec.io.sky (*module*), 84
desispec.io.spectra (*module*), 84
desispec.io.util (*module*), 86
desispec.io.xytraceset (*module*), 91
desispec.linalg (*module*), 91
desispec.log (*module*), 92
desispec.maskbits (*module*), 92
desispec.parallel (*module*), 93
desispec.pipeline (*module*), 95
desispec.pipeline.control (*module*), 95
desispec.pipeline.db (*module*), 101
desispec.pipeline.defs (*module*), 105
desispec.pipeline.prod (*module*), 106

desispec.pipeline.run (*module*), 107
desispec.pipeline.scriptgen (*module*), 109
desispec.pipeline.tasks (*module*), 111
desispec.pipeline.tasks.base (*module*), 111
desispec.pipeline.tasks.cframe (*module*), 115
desispec.pipeline.tasks.extract (*module*), 116
desispec.pipeline.tasks.fiberflat (*module*), 116
desispec.pipeline.tasks.fiberflatnight (*module*), 116
desispec.pipeline.tasks.fibermap (*module*), 117
desispec.pipeline.tasks.fluxcalib (*module*), 117
desispec.pipeline.tasks.preproc (*module*), 117
desispec.pipeline.tasks.psf (*module*), 118
desispec.pipeline.tasks.psfnight (*module*), 118
desispec.pipeline.tasks.qadata (*module*), 119
desispec.pipeline.tasks.rawdata (*module*), 119
desispec.pipeline.tasks.redshift (*module*), 120
desispec.pipeline.tasks.sky (*module*), 120
desispec.pipeline.tasks.spectra (*module*), 121
desispec.pipeline.tasks.starfit (*module*), 121
desispec.pipeline.tasks.traceshift (*module*), 122
desispec.pixgroup (*module*), 122
desispec.preproc (*module*), 124
desispec.qa (*module*), 128
desispec.qa.html (*module*), 128
desispec.qa.qa_brick (*module*), 129
desispec.qa.qa_exposure (*module*), 129
desispec.qa.qa_frame (*module*), 129
desispec.qa.qa_multiexp (*module*), 130
desispec.qa.qa_night (*module*), 130
desispec.qa.qa_plots (*module*), 130
desispec.qa.qa_plots_ql (*module*), 133
desispec.qa.qa_prod (*module*), 135
desispec.qa.qa_quicklook (*module*), 135
desispec.qa.qalib (*module*), 135
desispec.qa.utils (*module*), 138
desispec.qproc (*module*), 139
desispec.qproc.io (*module*), 139
desispec.qproc.qextract (*module*), 139
desispec.qproc.qfiberflat (*module*), 140
desispec.qproc.qframe (*module*), 140
desispec.qproc.qsky (*module*), 140
desispec.quicklook (*module*), 140
desispec.quicklook.arcprocess (*module*), 140
desispec.quicklook.merger (*module*), 140
desispec.quicklook.palib (*module*), 140
desispec.quicklook.pas (*module*), 141
desispec.quicklook.procalgs (*module*), 141
desispec.quicklook.qas (*module*), 142
desispec.quicklook.ql_plotlib (*module*), 144
desispec.quicklook.qlboxcar (*module*), 143
desispec.quicklook.qlconfig (*module*), 143
desispec.quicklook.qlexceptions (*module*), 144
desispec.quicklook.qlheartbeat (*module*), 144
desispec.quicklook.qllogger (*module*), 144
desispec.quicklook.qlpsf (*module*), 145
desispec.quicklook.qlresolution (*module*), 145
desispec.quicklook.quickfiberflat (*module*), 146
desispec.quicklook.quicklook (*module*), 146
desispec.quicklook.quicksky (*module*), 146
desispec.resolution (*module*), 147
desispec.scripts (*module*), 148
desispec.scripts.average_fiberflat (*module*), 148
desispec.scripts.bootcalib (*module*), 148
desispec.scripts.extract (*module*), 148
desispec.scripts.fiberflat (*module*), 148
desispec.scripts.fluxcalibration (*module*), 148
desispec.scripts.group_spectra (*module*), 148
desispec.scripts.mergebundles (*module*), 148
desispec.scripts.night (*module*), 148
desispec.scripts.pipe (*module*), 148
desispec.scripts.pipe_exec (*module*), 148
desispec.scripts.preproc (*module*), 148
desispec.scripts.procexp (*module*), 149
desispec.scripts.qa_exposure (*module*), 149
desispec.scripts.qa_frame (*module*), 149
desispec.scripts.qa_prod (*module*), 149
desispec.scripts.quicklook (*module*), 149
desispec.scripts.rejectcosmics (*module*), 150
desispec.scripts.sky (*module*), 150
desispec.scripts.skysubresid (*module*), 150
desispec.scripts.specex (*module*), 150
desispec.scripts.specscore (*module*), 150
desispec.scripts.stdstars (*module*), 150

desispec.scripts.trace_shifts (*module*), 151
desispec.scripts.update_spectra (*module*), 151
desispec.sky (*module*), 151
desispec.specscore (*module*), 153
desispec.spectra (*module*), 154
desispec.trace_shifts (*module*), 156
desispec.util (*module*), 162
desispec.workflow.schedule (*module*), 165
desispec.xytraceset (*module*), 165
difference_camwords() (*in module desispec.io.util*), 88
dist_balanced() (*in module desispec.parallel*), 93
dist_discrete() (*in module desispec.parallel*), 93
dist_discrete_all() (*in module desispec.parallel*), 93
dist_uniform() (*in module desispec.parallel*), 94
do_boxcar() (*desispec.quicklook.procalgs.BoxcarExtract method*), 141
do_boxcar() (*in module desispec.quicklook.qlboxcar*), 143
download() (*in module desispec.io.download*), 73
dry_run() (*in module desispec.pipeline.run*), 107
dryrun() (*in module desispec.pipeline.control*), 97
dump_job_env() (*in module desispec.pipeline.scriptgen*), 111
dump_pa() (*desispec.quicklook.qlconfig.Config method*), 143
dump_qa() (*desispec.quicklook.qlconfig.Config method*), 143

E

empty_fibermap() (*in module desispec.io.fibermap*), 74
environment variable
DESI_SPECTRO_CALIB, 38
expand_config() (*desispec.quicklook.qlconfig.Config method*), 143
exposure_fiberflat() (*in module desispec.qa.qa_plots*), 130
exposure_fluxcalib() (*in module desispec.qa.qa_plots*), 130
exposure_map() (*in module desispec.qa.qa_plots*), 130
exposure_s2n() (*in module desispec.qa.qa_plots*), 130
ExposureFlavor (*class in desispec.database.metadata*), 60
Extract_QP (*class in desispec.quicklook.procalgs*), 142
extract_sngfibers_gaussianpsf() (*in module desispec.bootcalib*), 49
Extraction_2d (*class in desispec.quicklook.procalgs*), 142

F

faflavor2program() (*in module desispec.io.meta*), 78
fast_resample_spectra() (*in module desispec.coaddition*), 57
fiber_gauss_new() (*in module desispec.bootcalib*), 50
fiber_gauss_old() (*in module desispec.bootcalib*), 50
FiberAssign (*class in desispec.database.redshift*), 62
fibermap2tilepix() (*in module desispec.pixgroup*), 123
fibermap_new2old() (*in module desispec.io.fibermap*), 74
fiducialregion() (*in module desispec.qa.qalib*), 136
find_arc_lines() (*in module desispec.bootcalib*), 50
find_exposure_night() (*in module desispec.io.meta*), 78
find_fiber_peaks() (*in module desispec.bootcalib*), 50
find_fiberassign_file() (*in module desispec.io.fibermap*), 75
findcalibfile() (*in module desispec.calibfinder*), 55
findfile() (*in module desispec.io.meta*), 78
finish() (*in module desispec.qa.html*), 129
fit_traces() (*in module desispec.bootcalib*), 51
fitsheader() (*in module desispec.io.util*), 88
fix_ycoeff_outliers() (*in module desispec.bootcalib*), 51
Flexure (*class in desispec.quicklook.procalgs*), 142
Frame (*class in desispec.database.metadata*), 60
frame_fiberflat() (*in module desispec.qa.qa_plots*), 130
frame_fluxcalib() (*in module desispec.qa.qa_plots*), 131
frame_s2n() (*in module desispec.qa.qa_plots*), 131
frame_skyres() (*in module desispec.qa.qa_plots*), 131
FrameLite (*class in desispec.pixgroup*), 122
frames2spectra() (*in module desispec.pixgroup*), 123
FrameStatus (*class in desispec.database.metadata*), 60
ftype (*desispec.spectra.Spectra attribute*), 155

G

gauss() (*in module desispec.qa.qalib*), 136

gen_scripts() (*in module desispec.pipeline.control*), 97
get_all_tiles() (*in module desispec.database.metadata*), 61
get_amp_ids() (*in module desispec.preproc*), 125
get_calibration_image() (*in module desispec.preproc*), 125
get_chan_spec_exp() (*in module desispec.quicklook.quicklook*), 146
get_channel_clrs() (*in module desispec.qa.qa_plots*), 131
get_default_config() (*desispec.quicklook.pas.PipelineAlg method*), 141
get_default_config() (*desispec.quicklook.procalgs.BoxcarExtract method*), 142
get_default_config() (*desispec.quicklook.procalgs.Extract_QP method*), 142
get_exp2healpix_map() (*in module desispec.pixgroup*), 123
get_exposures() (*in module desispec.io.meta*), 79
get_files() (*in module desispec.io.meta*), 79
get_frame() (*in module desispec.qa.qa_quicklook*), 135
get_gaia_ab_correction() (*in module desispec.scripts.stdstars*), 150
get_image() (*in module desispec.qa.qa_quicklook*), 135
get_inputs() (*in module desispec.qa.qa_quicklook*), 135
get_logger() (*in module desispec.log*), 92
get_magnitude() (*in module desispec.scripts.stdstars*), 150
get_nights() (*in module desispec.io.meta*), 80
get_options() (*in module desispec.database.redshift*), 62
get_pipe_database() (*in module desispec.io.meta*), 80
get_pipe_logdir() (*in module desispec.io.meta*), 80
get_pipe_nightdir() (*in module desispec.io.meta*), 80
get_pipe_pixeldir() (*in module desispec.io.meta*), 80
get_pipe_rundir() (*in module desispec.io.meta*), 80
get_pipe_scriptdir() (*in module desispec.io.meta*), 80
get_raw_files() (*in module desispec.io.meta*), 80
get_reduced_frames() (*in module desispec.io.meta*), 81
get_resampling_matrix() (*in module desispec.coaddition*), 57
get_resolution() (*in module desispec.quicklook.palib*), 140
Get_RMS (*class in desispec.qa.qa_quicklook*), 135
get_skyres() (*in module desispec.qa.utils*), 138
get_speclog() (*in module desispec.io.util*), 88
get_states() (*desispec.pipeline.db.DataBase method*), 102
get_states_type() (*desispec.pipeline.db.DataBase method*), 102
get_sty_otype() (*in module desispec.qa.qa_plots*), 131
get_submitted() (*desispec.pipeline.db.DataBase method*), 102
get_tasks() (*in module desispec.pipeline.control*), 98
get_tasks_type() (*in module desispec.pipeline.control*), 98
get_tempfilename() (*in module desispec.io.util*), 88
getready() (*desispec.pipeline.db.DataBase method*), 102
getready() (*desispec.pipeline.tasks.base.BaseTask method*), 112
getready() (*desispec.pipeline.tasks.fiberflatnight.TaskFiberflatNight method*), 117
getready() (*desispec.pipeline.tasks.psfnight.TaskPSFNight method*), 119
getready() (*in module desispec.pipeline.control*), 99
getrms() (*in module desispec.qa.qalib*), 136
gpu_specter_check_input_options() (*in module desispec.scripts.extract*), 148

H

header() (*in module desispec.qa.html*), 129
header2night() (*in module desispec.util*), 162
header2wave() (*in module desispec.io.util*), 89
healpix_degrade_fixed() (*in module desispec.util*), 162
healpix_subdirectory() (*in module desispec.io.util*), 89

I

id_arc_lines_using_triplets() (*in module desispec.bootcalib*), 51
initdb() (*desispec.pipeline.db.DataBasePostgres method*), 104
initdb() (*desispec.pipeline.db.DataBaseSqlite method*), 104
Initialize (*class in desispec.quicklook.procalgs*), 142
insert() (*desispec.pipeline.tasks.base.BaseTask method*), 112
Integrate_Spec (*class in desispec.qa.qa_quicklook*), 135

integrate_spec() (*in module desispec.qa.qalib*), 136
interpolate_on_parameter_grid() (*in module desispec.fluxcalibration*), 68
io_qa() (*desispec.quicklook.qlconfig.Config method*), 143
io_qa_pa() (*desispec.quicklook.qlconfig.Config method*), 143
is_svn_current() (*in module desispec.io.util*), 89
isStdStar() (*in module desispec.fluxcalibration*), 69
itemindices() (*in module desispec.util*), 162
iterfiles() (*in module desispec.io.util*), 89

L

legx() (*in module desispec.trace_shifts*), 160
load_arcline_list() (*in module desispec.bootcalib*), 51
load_data() (*in module desispec.database.metadata*), 61
load_db() (*in module desispec.pipeline.db*), 105
load_fiberassign() (*in module desispec.database.redshift*), 62
load_file() (*in module desispec.database.redshift*), 63
load_filter() (*in module desispec.io.filters*), 75
load_gaia_filter() (*in module desispec.io.filters*), 75
load_gdarc_lines() (*in module desispec.bootcalib*), 52
load_legacy_survey_filter() (*in module desispec.io.filters*), 76
load_parse_dict() (*in module desispec.bootcalib*), 52
load_prod() (*in module desispec.pipeline.prod*), 106
load_qa_brick() (*in module desispec.io.qa*), 82
load_qa_frame() (*in module desispec.io.qa*), 82
load_qa_multiexp() (*in module desispec.io.qa*), 82
load_redrock() (*in module desispec.database.redshift*), 63
load_simulated_data() (*in module desispec.database.metadata*), 62

M

main() (*in module desispec.database.metadata*), 62
main() (*in module desispec.database.redshift*), 63
main() (*in module desispec.scripts.stdstars*), 150
make_exposure() (*in module desispec.qa.html*), 129
make_exposures() (*in module desispec.qa.html*), 129
makepath() (*in module desispec.io.util*), 89
mapkeywords() (*in module desispec.quicklook.quicklook*), 146
mask32() (*in module desispec.util*), 163
match_templates() (*in module desispec.fluxcalibration*), 69
mean_psf() (*in module desispec.scripts.specex*), 150
merge_psf() (*in module desispec.scripts.specex*), 150
mjd2night() (*in module desispec.util*), 163
MonitoringAlg (*class in desispec.quicklook.qas*), 142
monomials() (*in module desispec.trace_shifts*), 160
mpi_count_failures() (*in module desispec.util*), 163

N

name_join() (*desispec.pipeline.tasks.base.BaseTask method*), 112
name_split() (*desispec.pipeline.tasks.base.BaseTask method*), 112
native_endian() (*in module desispec.io.util*), 89
nersc_job() (*in module desispec.pipeline.scriptgen*), 111
Night (*class in desispec.database.metadata*), 60
night2ymd() (*in module desispec.util*), 163
normalize_templates() (*in module desispec.fluxcalibration*), 69
num_spectra() (*desispec.pixgroup.SpectraLite method*), 122
num_spectra() (*desispec.spectra.Spectra method*), 155
num_targets() (*desispec.pixgroup.SpectraLite method*), 122
num_targets() (*desispec.spectra.Spectra method*), 155
numba_mean() (*in module desispec.preproc*), 126

O

ObsList (*class in desispec.database.redshift*), 62
offset() (*desispec.database.metadata.Tile method*), 60
option_list() (*in module desispec.util*), 163
ordered_unique() (*in module desispec.util*), 163
orig_SNRFit() (*in module desispec.qa.qalib*), 137
overlapping_bricks() (*desispec.database.metadata.Tile method*), 61

P

paargs (*desispec.quicklook.qlconfig.Config attribute*), 143
Palist (*class in desispec.quicklook.qlconfig*), 143
palist (*desispec.quicklook.qlconfig.Config attribute*), 143
ParameterException, 144
parse() (*in module desispec.scripts.quicklook*), 149
parse_badamps() (*in module desispec.io.util*), 89
parse_cameras() (*in module desispec.io.util*), 90
parse_date_obs() (*in module desispec.calibfinder*), 56

parse_fibers() (*in module desispec.util*), 163
parse_int_args() (*in module desispec.util*), 164
parse_job_env() (*in module desispec.pipeline.scriptgen*), 111
parse_nist() (*in module desispec.bootcalib*), 52
parse_nist_tbl() (*in module desispec.bootcalib*), 52
parse_pgpass() (*in module desispec.database.util*), 64
parse_sec_keyword() (*in module desispec.preproc*), 126
paths() (*desispec.pipeline.tasks.base.BaseTask method*), 112
petals() (*desispec.database.metadata.Tile method*), 61
pid_exists() (*in module desispec.util*), 164
PipelineAlg (*class in desispec.quicklook.pas*), 141
plot_bias_overscan() (*in module desispec.qa.qa_plots_ql*), 134
plot_countpix() (*in module desispec.qa.qa_plots_ql*), 134
plot_countspectralbins() (*in module desispec.qa.qa_plots_ql*), 134
plot_lpolyhist() (*in module desispec.qa.qa_plots_ql*), 134
plot_RMS() (*in module desispec.qa.qa_plots_ql*), 133
plot_sky_continuum() (*in module desispec.qa.qa_plots_ql*), 134
plot_sky_peaks() (*in module desispec.qa.qa_plots_ql*), 134
plot_SNR() (*in module desispec.qa.qa_plots_ql*), 133
plot_XWSigma() (*in module desispec.qa.qa_plots_ql*), 134
polynomial_fit() (*in module desispec.trace_shifts*), 160
postprocessing() (*desispec.pipeline.tasks.base.BaseTask method*), 112
postprocessing() (*desispec.pipeline.tasks.cframe.TaskCFrame method*), 115
postprocessing() (*desispec.pipeline.tasks.extract.TaskExtract method*), 116
postprocessing() (*desispec.pipeline.tasks.fiberflat.TaskFiberflat method*), 116
postprocessing() (*desispec.pipeline.tasks.fiberflatnight.TaskFiberflatNight method*), 117
postprocessing() (*desispec.pipeline.tasks.fluxcalib.TaskFluxCalib method*), 117
postprocessing() (*desispec.pipeline.tasks.preproc.TaskPreproc method*), 118
postprocessing() (*desispec.pipeline.tasks.psf.TaskPSF method*), 118
postprocessing() (*desispec.pipeline.tasks.psfnight.TaskPSFNight method*), 119
postprocessing() (*desispec.pipeline.tasks.qadata.TaskQAData method*), 119
postprocessing() (*desispec.pipeline.tasks.sky.TaskSky method*), 121
postprocessing() (*desispec.pipeline.tasks.starfit.TaskStarFit method*), 122
postprocessing() (*desispec.pipeline.tasks.traceshift.TaskTraceShift method*), 122
Preproc (*class in desispec.quicklook.procalgs*), 142
preproc() (*in module desispec.preproc*), 126
preproc_file() (*in module desispec.scripts.preproc*), 149
process_arc() (*in module desispec.quicklook.arcprocess*), 140
prod_avg_s2n() (*in module desispec.qa.qa_plots*), 132
prod_channel_hist() (*in module desispec.qa.qa_plots*), 132
prod_options_name (*in module desispec.pipeline.defs*), 105
prod_time_series() (*in module desispec.qa.qa_plots*), 132
prod_ZP() (*in module desispec.qa.qa_plots*), 131
project() (*in module desispec.quicklook.palib*), 141
PSF (*class in desispec.quicklook.qlpsf*), 145

Q

q3c_index() (*in module desispec.database.redshift*), 64
qa_arc_spec() (*in module desispec.bootcalib*), 53
qa_fiber_arcrms() (*in module desispec.bootcalib*), 53
qa_fiber_dlamb() (*in module desispec.bootcalib*), 53
qa_fiber_Dx() (*in module desispec.bootcalib*), 53
qa_fiber_gauss() (*in module desispec.bootcalib*), 53
qa_fiber_peaks() (*in module desispec.bootcalib*), 53
qa_fiber_trace() (*in module desispec.bootcalib*), 54
qa_fiberflat() (*in module desispec.fiberflat*), 66

qa_fluxcalib() (*in module desispec.fluxcalibration*), 70
qa_skysub() (*in module desispec.sky*), 153
qafile_from_framefile() (*in module desispec.io.qa*), 82
qaframe_from_frame() (*in module desispec.qa.qa_frame*), 129
qalist (*desispec.quicklook.qlconfig.Config attribute*), 143
qaprod_root() (*in module desispec.io.meta*), 81
QASeverity (*class in desispec.quicklook.qas*), 142
ql_2dplot() (*in module desispec.quicklook.ql_plotlib*), 144
ql_3dplot() (*in module desispec.quicklook.ql_plotlib*), 144
ql_patchplot() (*in module desispec.quicklook.ql_plotlib*), 144
ql_qaplot() (*in module desispec.quicklook.ql_plotlib*), 145
QLLogger (*class in desispec.quicklook.qllogger*), 144
qproc_apply_fiberflat() (*in module desispec.qproc.qfiberflat*), 140
qproc_boxcar_extraction() (*in module desispec.qproc.qextract*), 139
qproc_compute_fiberflat() (*in module desispec.qproc.qfiberflat*), 140
qproc_sky_subtraction() (*in module desispec.qproc.qsky*), 140
QuickResolution (*class in desispec.quicklook.qlresolution*), 145

R

radius (*desispec.database.metadata.Tile attribute*), 61
rawdata_root() (*in module desispec.io.meta*), 81
read() (*desispec.pixgroup.FrameLite class method*), 122
read() (*desispec.pixgroup.SpectraLite class method*), 122
read_average_flux_calibration() (*in module desispec.io.fluxcalibration*), 76
read_bias() (*in module desispec.preproc*), 127
read_dark() (*in module desispec.preproc*), 128
read_fiberflat() (*in module desispec.io.fiberflat*), 73
read_fibermap() (*in module desispec.io.fibermap*), 75
read_flux_calibration() (*in module desispec.io.fluxcalibration*), 76
read_frame() (*in module desispec.io.frame*), 77
read_frame_as_spectra() (*in module desispec.io.spectra*), 84
read_image() (*in module desispec.io.image*), 78
read_mask() (*in module desispec.preproc*), 128
read_meta_frame() (*in module desispec.io.frame*), 77
read_params() (*in module desispec.io.params*), 81
read_pixflat() (*in module desispec.preproc*), 128
read_qa_brick() (*in module desispec.io.qa*), 82
read_qa_data() (*in module desispec.io.qa*), 82
read_qa_frame() (*in module desispec.io.qa*), 82
read_qframe() (*in module desispec.qproc.io*), 139
read_raw() (*in module desispec.io.raw*), 83
read_sky() (*in module desispec.io.sky*), 84
read_spectra() (*in module desispec.io.spectra*), 85
read_stdstar_models() (*in module desispec.io.fluxcalibration*), 76
read_stdstar_templates() (*in module desispec.io.fluxcalibration*), 76
read_tile_spectra() (*in module desispec.io.spectra*), 85
read_xytraceset() (*in module desispec.io.xytraceset*), 91
recompute_legendre_coefficients() (*in module desispec.trace_shifts*), 160
redshift_fit() (*in module desispec.fluxcalibration*), 70
reject_cosmic_rays() (*in module desispec.cosmics*), 59
reject_cosmic_rays_1d() (*in module desispec.cosmics*), 59
reject_cosmic_rays_ala_sdss() (*in module desispec.cosmics*), 59
reject_lines() (*in module desispec.bootcalib*), 54
replace_prefix() (*in module desispec.io.util*), 90
resample_boxcar_frame() (*in module desispec.trace_shifts*), 161
resample_flux() (*in module desispec.interpolation*), 72
resample_spec() (*in module desispec.quicklook.palib*), 141
resample_spectra_lin_or_log() (*in module desispec.coaddition*), 57
resample_template() (*in module desispec.fluxcalibration*), 70
Resolution (*class in desispec.resolution*), 147
ResolutionFit (*class in desispec.quicklook.procalgs*), 142
retrieve() (*desispec.pipeline.tasks.base.BaseTask method*), 112
run() (*desispec.pipeline.tasks.base.BaseTask method*), 113
run() (*in module desispec.pipeline.control*), 99
run() (*in module desispec.scripts.specex*), 150
run_and_update() (*desispec.pipeline.tasks.base.BaseTask method*), 113
run_and_update() (*desis-*

pec.pipeline.tasks.redshift.TaskRedshift method), 120
run_and_update() (*desispec.pipeline.tasks.spectra.TaskSpectra method*), 121
run_cli() (*desispec.pipeline.tasks.base.BaseTask method*), 113
run_defaults() (*desispec.pipeline.tasks.base.BaseTask method*), 113
run_dist() (*in module desispec.pipeline.run*), 108
run_max_mem_proc() (*desispec.pipeline.tasks.base.BaseTask method*), 114
run_max_mem_task() (*desispec.pipeline.tasks.base.BaseTask method*), 114
run_max_procs() (*desispec.pipeline.tasks.base.BaseTask method*), 114
run_scripts() (*in module desispec.pipeline.control*), 100
run_task() (*in module desispec.pipeline.run*), 108
run_task_list() (*in module desispec.pipeline.run*), 108
run_task_list_db() (*in module desispec.pipeline.run*), 109
run_task_simple() (*in module desispec.pipeline.run*), 109
run_time() (*desispec.pipeline.tasks.base.BaseTask method*), 114
runcmd() (*in module desispec.util*), 164
runpipeline() (*in module desispec.quicklook.quicklook*), 146

S

s2n_flux_astro() (*in module desispec.qa.qalib*), 137
s2n_funcs() (*in module desispec.qa.qalib*), 137
s2nfit() (*in module desispec.qa.qalib*), 138
SchemaMixin (*class in desispec.database.redshift*), 62
script() (*in module desispec.pipeline.control*), 100
script_bootcalib() (*in module desispec.bootcalib*), 54
search_for_framefile() (*in module desispec.io.frame*), 77
select() (*desispec.spectra.Spectra method*), 155
select_nights() (*in module desispec.pipeline.prod*), 106
set_states() (*desispec.pipeline.db.DataBase method*), 102
set_states_type() (*desispec.pipeline.db.DataBase method*), 103
set_submitted() (*desispec.pipeline.db.DataBase method*), 103
set_submitted_type() (*desispec.pipeline.db.DataBase method*), 103
setup_db() (*in module desispec.database.redshift*), 64
setup_pipeline() (*in module desispec.quicklook.quicklook*), 146
shift_ycoef_using_external_spectrum() (*in module desispec.trace_shifts*), 161
shorten_filename() (*in module desispec.io.meta*), 81
show_meta() (*in module desispec.qa.qa_plots*), 132
sigmas_from_arc() (*in module desispec.quicklook.arcprocess*), 140
SignalVsNoise() (*in module desispec.qa.qalib*), 135
simulate_frame() (*desispec.database.metadata.Tile method*), 61
Sky_Continuum (*class in desispec.qa.qa_quicklook*), 135
sky_continuum() (*in module desispec.qa.qalib*), 138
Sky_Peaks (*class in desispec.qa.qa_quicklook*), 135
Sky_Rband (*class in desispec.qa.qa_quicklook*), 135
sky_resid() (*in module desispec.qa.qalib*), 138
Sky_Residual (*class in desispec.qa.qa_quicklook*), 135
skyline_resid() (*in module desispec.qa.qa_plots*), 133
SkySub (*class in desispec.quicklook.procalgs*), 142
skysub_gauss() (*in module desispec.qa.qa_plots*), 133
SkySub_QL (*class in desispec.quicklook.procalgs*), 142
SkySub_QP (*class in desispec.quicklook.procalgs*), 142
skysub_resid_dual() (*in module desispec.qa.qa_plots*), 133
skysub_resid_series() (*in module desispec.qa.qa_plots*), 133
slice_fidboundary() (*in module desispec.qa.qalib*), 138
sm2sp() (*in module desispec.calibfinder*), 56
SN_ratio() (*in module desispec.qa.qalib*), 135
sp2sm() (*in module desispec.calibfinder*), 56
specprod_root() (*in module desispec.io.meta*), 81
Spectra (*class in desispec.spectra*), 154
SpectraLite (*class in desispec.pixgroup*), 122
spectroperf_resample_spectra() (*in module desispec.coaddition*), 58
spline_fit() (*in module desispec.linalg*), 92
sprun() (*in module desispec.util*), 164
stack() (*in module desispec.spectra*), 156
state_colors (*in module desispec.pipeline.defs*), 105
state_get() (*desispec.pipeline.tasks.base.BaseTask method*), 114
state_set() (*desispec.pipeline.tasks.base.BaseTask method*), 115

Status (*class in desispec.database.metadata*), 60
status() (*in module desispec.pipeline.control*), 101
stdouterr_redirected() (*in module desispec.parallel*), 94
subtract_peramp_overscan() (*in module desispec.preproc*), 128
subtract_sky() (*in module desispec.quicklook.quicksky*), 146
subtract_sky() (*in module desispec.sky*), 153
sync() (*desispec.pipeline.db.DataBase method*), 103
sync() (*in module desispec.pipeline.control*), 101

T

take_turns() (*in module desispec.parallel*), 94
Target (*class in desispec.database.redshift*), 62
target_ids() (*desispec.pixgroup.SpectraLite method*), 122
target_ids() (*desispec.spectra.Spectra method*), 156
task_name_sep (*in module desispec.pipeline.defs*), 105
task_read() (*in module desispec.pipeline.prod*), 106
task_sort() (*in module desispec.pipeline.db*), 105
task_states (*in module desispec.pipeline.defs*), 105
task_type() (*in module desispec.pipeline.tasks.base*), 115
task_write() (*in module desispec.pipeline.prod*), 106
TaskCFrame (*class in desispec.pipeline.tasks.cframe*), 115
TaskExtract (*class in desispec.pipeline.tasks.extract*), 116
TaskFiberflat (*class in desispec.pipeline.tasks.fiberflat*), 116
TaskFiberflatNight (*class in desispec.pipeline.tasks.fiberflatnight*), 116
TaskFibermap (*class in desispec.pipeline.tasks.fibermap*), 117
TaskFluxCalib (*class in desispec.pipeline.tasks.fluxcalib*), 117
TaskPreproc (*class in desispec.pipeline.tasks.preproc*), 117
TaskPSF (*class in desispec.pipeline.tasks.psf*), 118
TaskPSFNight (*class in desispec.pipeline.tasks.psfnight*), 118
TaskQAData (*class in desispec.pipeline.tasks.qadata*), 119
TaskRawdata (*class in desispec.pipeline.tasks.rawdata*), 119
TaskRedshift (*class in desispec.pipeline.tasks.redshift*), 120
tasks() (*in module desispec.pipeline.control*), 101
TaskSky (*class in desispec.pipeline.tasks.sky*), 120
TaskSpectra (*class in desispec.pipeline.tasks.spectra*), 121
TaskStarFit (*class in desispec.pipeline.tasks.starfit*), 121
TaskTraceShift (*class in desispec.pipeline.tasks.traceshift*), 122
Tile (*class in desispec.database.metadata*), 60
TimeoutError, 107
to_fits_array() (*desispec.resolution.Resolution method*), 147
toplevel() (*in module desispec.qa.html*), 129
trace_crude_init() (*in module desispec.bootcalib*), 54
trace_fweight() (*in module desispec.bootcalib*), 55
Trace_Shifts (*class in desispec.qa.qa_quicklook*), 135
Truth (*class in desispec.database.redshift*), 62

U

update() (*desispec.pipeline.db.DataBase method*), 103
update() (*desispec.spectra.Spectra method*), 156
update() (*in module desispec.pipeline.control*), 101
update_frame_cache() (*in module desispec.pixgroup*), 123
update_prod() (*in module desispec.pipeline.prod*), 106
update_truth() (*in module desispec.database.redshift*), 64
use_mpi() (*in module desispec.parallel*), 94

V

validate_badamps() (*in module desispec.io.util*), 90
validate_night() (*in module desispec.io.meta*), 81

W

wavelength() (*desispec.quicklook.qlpsf.PSF method*), 145
wavelength_grid() (*desispec.spectra.Spectra method*), 156
weighted_partition() (*in module desispec.parallel*), 94
write() (*desispec.pixgroup.SpectraLite method*), 123
write_average_flux_calibration() (*in module desispec.io.fluxcalibration*), 76
write_bintable() (*in module desispec.io.util*), 90
write_fiberflat() (*in module desispec.io.fiberflat*), 73
write_fibermap() (*in module desispec.io.fibermap*), 75
write_flux_calibration() (*in module desispec.io.fluxcalibration*), 76
write_frame() (*in module desispec.io.frame*), 77

write_image() (*in module desispec.io.image*), 78
write_psf() (*in module desispec.bootcalib*), 55
write_psffile() (*in module desispec.quicklook.arcprocess*), 140
write_qa_brick() (*in module desispec.io.qa*), 82
write_qa_exposure() (*in module desispec.io.qa*), 82
write_qa_frame() (*in module desispec.io.qa*), 83
write_qa_multiexp() (*in module desispec.io.qa*), 83
write_qa_ql() (*in module desispec.io.qa*), 83
write_qframe() (*in module desispec.qproc.io*), 139
write_raw() (*in module desispec.io.raw*), 84
write_sky() (*in module desispec.io.sky*), 84
write_spectra() (*in module desispec.io.spectra*), 85
write_stdstar_models() (*in module desispec.io.fluxcalibration*), 76
write_traces_in_psf() (*in module desispec.trace_shifts*), 162
write_xytraceset() (*in module desispec.io.xytraceset*), 91

X

x() (*desispec.quicklook.qlpsf.PSF method*), 145

Y

y() (*desispec.quicklook.qlpsf.PSF method*), 145
yaml_read() (*in module desispec.pipeline.prod*), 107
yaml_write() (*in module desispec.pipeline.prod*), 107
ymd2night() (*in module desispec.util*), 165

Z

ZCat (*class in desispec.database.redshift*), 62
ZP_from_calib() (*in module desispec.fluxcalibration*), 66