NOAO SDM Document PL028

Operator's Guide to the DECam Community Pipeline

F. Valdes[1]

National Optical Astronomy Observatories
Science Data Management

V1 August 25, 2020

[1]NSF's OIR Lab/Community Science Data Center, P.O. Box 26732, Tucson, AZ 85732

Table of Contents

1 Introduction

The DECam Community Pipeline produces data products from the raw exposure files taken by observers with the camera. The data products apply basic flux and astrometric calibrations and may included stacked (aka coadded) full field combinations as appropriate and selected by the operator. This guide presents overview and procedural information for the operator of the pipeline. While the overiew if general, the operational information is specific to the configuration at Community Science Data Center (CSDC) of NSF's Optical/Infrared Astronomy Research Laboratory (NSF's OIR Lab). The description is also specific to the most standard operation of the CP but with indications of optional procedures that might be of use.

This document begins with overviews of the workflow organization and of the CP architecture. It is then followed by the various areas of operation ordered roughly in the order in which the operator intereacts with the pipeline. The commands the operator (or pipeline) executes are described in some detail but the ultimate detail lies in the command's usage guide and help topic.

2 The Pipeline Architecture

The pipeline operator needs to understand the basic structure and logic of the pipeline workflow/orchestration system, though not its internal details. A *pipeline orchestrator* is what directs the sequence of steps or stages executed and provides error handling and monitoring. For high data volume applications with a lot of data parallelism, such as for DECam exposures, an orchestration system generally provides distributed parallel processing execution. The Community Pipeline orchestrator uses the *NOAO High-Performance Pipelines System (NHPPS)* which embodies these features with ease of configuration and workflow defintion on a cluster of hosts.

The NHPPS is described in detail in **?**. This is slightly out of date but is still fairly relevant. In this section of the guide we provide a high level overview appropriate for an operator.

We must first consider the organization and logic of an NHPPS pipeline application before we discuss the software components. The term *pipeline application* is used to distinguish the CP from elements of the architecture which are called *pipelines*. In other words, a pipeline application, like the CP, is built from a set of pipeline elements, each of which operates on a particular type of input, called a *dataset* and does particular aspects of the processing on the dataset. This is much like subroutines or methods in a program. Admittedly, the overloaded use of the term pipeline is a little confusing. But it is important the operator understands that the CP has a number of things called pipelines since this is vital to the discussion, description, and operation of the CP.

The CP under the NHPPS takes an input DECam dataset and orchestrates it through the various pipelines. A pipeline may decompose the dataset into subdatasets to be operated upon, possibly in parallel, by other pipelines. These pipelines will be running on multiple hosts. As a *child pipeline* finishes it passes control back to the *parent pipeline*. When there are multiple pieces of the parent dataset being operated upon by multiple child pipelines the parent pipeline keeps track of the returns until all of the pieces have been returned. This structure of parallelizing is called a *scatter/gather strategy*.

A pipeline is composed of a number of discrete steps called *stages* or *modules*. In a stage a particular host command is executed by NHPPS and then it waits for the command to finish with some exit status which may be success or an error of some type. Continuing with the program analogy, the stages are like the statements in a subroutine.

Unlike a traditional program workflow, the stages are not necessarily executed in some linear order where control is passed step by step. Instead each stage excutes when certain *trigger conditions* are met. This is called an *event-driven strategy*. While NHPPS support a variety of events, the CP is makes use of two types – *trigger files* and *status flags*. The status flags are often related to the exit statuses noted previously along with a status that can be roughly thought of as not having been executed. We will call this the underscore condition, _, for reasons to become apparent later. In the event-driven approach _ is just one of the conditions required for a stage to execute. The other conditions are usually the successful complete of one or more dependent stages. Since most often the stages are displayed in an order where the preceding stage has to have completed the pipeline execution appears somewhat linear but technically that is an illusion.

NHPPS keeps track of status flags for each stage in each pipeline running on each node on each dataset with a structure called the *blackboard*. This is a fundamental operations concept since the

operator tracks the progress of the pipeline, including any errors or other conditions that might halt processing, with a visual representation of the blackboard. The operator can also modify the processing workflow by directly interacting with the blackboard to affect the trigger conditions.

In addition, to status flags the blackboard keeps track of the number of pieces of work distributed by a parent pipeline to child pipelines. As each child dataset is completed a trigger is returned to the wait stage of the parent and the blackboard count is decremented. Monitoring these counts is an important aspect of the operations for two reasons. One is that if the count never reaches zero this means something failed. Secondly, by monitoring the count for the parent is more efficient than monitoring the, possibly large, set of children pipelines spread across many hosts.

Another important concept that makes it easier to understand and remember pipelines and stages in the workflow of the CP are conventions that are standardized across pipelines. Some common types of stages are those for triggering of new datasets, completion of datasets, and the call and return of child datasets. In naming conventions, all pipelines have three character mnemonic names and all stages in a pipeline begin with the pipeline name.

2.1 The NHPPS Software Components

The NHPPS has a few types of servers, some of which have many instances across the cluster of hosts, that are started by the operator (see §5) and then remain running for long periods of time. The two key ones to understand about how the pipeline works are the *node managers* and especially the *pipeline managers*.

There is one node manager for each host. If a host does not have a node manager running then that host will not be used. The node manager starts and stops pipeline managers on that node. There is one pipeline manager for each type configured to run on the node. Each pipeline manager is configured to run up to some number of instances of each stage at the same time. Note this limit is for a stage, e.g. 4 instances of omiwcs, and does not limit the number of stages, say 5 distinct stages each with up to 4 instances of that stage, in the pipeline running at the same time.

Throttling of the pipelines is important. So in addition to the configuration just mentioned, there is a configuration parameter that limits how many child datasets may be created by a parent on each node. Remaining child datasets are staged and submitted to a node as a child pipeline dataset returns from that node. This throttles the work and also has the consequence that slower nodes may ultimately process fewer of the child datasets.

The throttling configuration is set by the CP developer. The operator may learn to adjust these with time and also has some control with an operator command to trigger additional child datasets before the datasets return. These are more advanced capabilities.

As noted in the last section, the NHPPS blackboard is a logical component of the system. Each pipeline manager keeps their own piece of the blackboard. All or subsets of these can be queried and collected for display which is what the pipeline blackboard monitor tool does. Similarly, operator tools can change status flags in various combinations from a single flag on a single node for a single node to all flags for a stage, dataset, or pipeline.

3 The CP Account and Environment

The Community Pipeline is operated from the account **deccp** which is present on all pipeline hosts in the pipeline cluster. The operator normally is logged into one host (currently **dec01**), referred to as the data manager, from several terminal windows. The windows are used for running pipeline commands or monitoring. Logging onto other hosts is usually for diagnosing problems.

The pipeline account provides a standard environment with environment variables, executable paths, data paths, and aliases. The current pipeline environment is based on the **tcsh** shell. It is configured by the **.cshrc** (and files sourced from it) and **aliases.csh** files. The operator may need to modify a few things in the environment for the operation of the pipeline. But also, the operator can customize the environment for their ease of use; in particular by aliases and **cdpath**. The latter allows changing to heavily used directories without typing a lot of paths. In this guide directories in the current **cdpath** are shown without paths.

Many of the operator commands are built in a hierarchy from low level commands to csh scripts to aliases.

In summary, the operator is encouraged to explore and understand the environment, customize aliases and **cdpath**, and study some of the csh script commands. This allows understanding the lower level pipeline commands which may not be explicitly covered in this operator's guide.

Figure 1: High level workflow overview.

- trigger
- stage
- group
- calibration data
- science data
  - exposure flux/astrometric
  - exposure projection
  - stack
- save results
- review
- archive

4 Workflow Overview

Data to be pipeline processed is called a dataset. A dataset is most typically all the exposures from an observing run by a particular principal investigator (PI) with a particular proposal identification (PROPID). However, other common datasets are a single night or a specific list of exposures which are, possibly, a subset of a run or spread out across time. Basically, a dataset if a flexibly defined set of exposures.

The concept of a dataset as a list of items is fundamental to the architecture. The workflow forms subsets of the exposures or components of the exposures which then go into sub-workflows in increasingly finer detail. Each of these datasets are operated on by a sub-pipeline on a computational host. By subdividing datasets this allows multiple hosts and pipelines to be processing in a parallel and distributed fashion. The orchestration architecture takes care of assigning and tracking all these parallel datasets. See 2 for a more detail description of this workflow orchestration.

The basic stages of the workflow are shown in figure 2. Triggering is the term for telling the pipeline to process a dataset. The exposures in the dataset are staged from the archive and initially grouped by type (calibration vs science) and filter. More rarely the grouping may be further divided into nights. Also at this point certain exposures are ignored based on missing keywords, bad format, being in an ignore operator file or having particular words in the observer's titles (OBJECT keyword). These special, case insensitive, words are shown in figure**??**. Note that "test" is allowed.

The zero (aka bias) and dome flat groups are processed first by night. However, the pipeline will check if there is a version of these already in the calibration library (see 2) and skip this part of the workflow. This means, as a detail, that if the operator want to force a new calibration to be created steps must be taken to override this behavior. The zero dataset (only one per night) is processed first and the result added to the calibration library for future use; typically immediately on the current dataset being processed. The dome flat exposures, divided into separate filter datasets, and

Table 1: Table of title words for which exposures are ignored.

focus	case insensitive
donut	case insensitive
junk	case insensitive
pointing	case insensitive
aos	at beginning of title, case insensitive
Amap	at beginning of title
PPMap	at beginning of title

processed next and added to the calibration library.

After the calibration data has been processed into calibration files the science images (having OBSTYPE of "object" or "standard") are processed by filter. There are three logical components of the workflow. Individual flux and astrometric calibration, resampling (aka reprojecting) to a uniform orientation and sampling, and stacking by pointings with multiple overlapping exposures (dithered or not).

The calibration and science processing (by filter) end with moving the results to a final repository area. After data is moved the operator submits the data products to the archive which are then deleted. However, the final repository also holds the processing logs and operator review graphics which are retained essentially indefinitely and are review at the time or if questions come up.

5 Standard Operations

In this section the standard operation mode is presented. This assumes nothing fails and that the processing queue (see **??** has been set up.

Data to be pipeline processed is called a dataset. A dataset is most typically all the exposures from an observing run by a particular principal investigator (PI) with a particular proposal identification (PROPID). However, other common datasets are a single night or a specific list of exposures which are, possibly, a subset of a run or spread out across time. Basically, a dataset if a flexibly defined set of exposures.

The concept of a dataset as a list of items is fundamental to the architecture. The workflow forms subsets of the exposures or components of the exposures which then go into sub-workflows in increasingly finer detail. Each of these datasets are operated on by a sub-pipeline on a computational host. By subdividing datasets this allows multiple hosts and pipelines to be processing in a parallel and distributed fashion. The orchestration architecture takes care of assigning and tracking all these parallel datasets. See 2 for a more detail description of this workflow orchestration.

Figure 2: High level workflow overview.

- trigger
- stage
- group
- calibration data
- science data
  - exposure flux/astrometric
  - exposure projection
  - stack
- save results
- review
- archive

The basic stages of the workflow are shown in figure 2. Triggering is the term for telling the pipeline to process a dataset. The exposures in the dataset are staged from the archive and initially grouped by type (calibration vs science) and filter. More rarely the grouping may be further divided into nights. Also at this point certain exposures are ignored based on missing keywords, bad format, being in an ignore operator file or having particular words in the observer's titles (OBJECT keyword). These special, case insensitive, words are shown in figure**??**. Note that "test" is allowed.

The zero (aka bias) and dome flat groups are processed first by night. However, the pipeline will check if there is a version of these already in the calibration library (see 2) and skip this part of the workflow. This means, as a detail, that if the operator want to force a new calibration to be created steps must be taken to override this behavior. The zero dataset (only one per night) is processed

Table 2: Table of title words for which exposures are ignored.

focus	case insensitive
donut	case insensitive
junk	case insensitive
pointing	case insensitive
aos	at beginning of title, case insensitive
Amap	at beginning of title
PPMap	at beginning of title

first and the result added to the calibration library for future use; typically immediately on the current dataset being processed. The dome flat exposures, divided into separate filter datasets, and processed next and added to the calibration library.

After the calibration data has been processed into calibration files the science images (having OBSTYPE of "object" or "standard") are processed by filter. There are three logical components of the workflow. Individual flux and astrometric calibration, resampling (aka reprojecting) to a uniform orientation and sampling, and stacking by pointings with multiple overlapping exposures (dithered or not).

The calibration and science processing (by filter) end with moving the results to a final repository area. After data is moved the operator submits the data products to the archive which are then deleted. However, the final repository also holds the processing logs and operator review graphics which are retained essentially indefinitely and are review at the time or if questions come up.

6 Starting and Stopping the Pipeline Application

The Community Pipeline infrastructure is generally always active and either waiting for a trigger or processing. The operator typically has one or more windows continuously open on their desktop to monitor and interact with the pipeline through the operator tools provided. Naturally, however, the infrastructure has to occasionally be restarted, such as after an event that causes the pipeline hosts to be rebooted. This section describes how to setup the pipeline to be ready to process and be monitored. For completeness, stopping the pipeline gracefully is also covered.

The first step is to login to the pipeline operator account deccp on the operator host dec01. If you are unsure about the status of the pipeline and its hosts type **`plstatus`** . This will check all the hosts identified by the environment variable **`PROCNODES`** . Figures 3 and 4 show example outputs when the pipeline is not running and when it is. Connexted tests is the host up and NodeMgr tests if the host is running the pipeline infrastructure. The available pipelines are the subpipelines available on the host.

If **`plstatus`** shows the nodes are all connected and running pipelines then everything is ready to process data. If it shows the hosts are up but no pipelines then the next step is **`plrun`** . Figure 5 shows the output from the command. It identifies the pipeline application (the CP), the operator node (dec01 in this case), and the processing nodes (**`PROCNODES`**). The steps it follows are to first remove any old data (see help for options), start the various servers need by NHPPS (name server, directory server, and the node managers), and finally start the pipeline manager for each subpipeline. WHen everything is successfully start the command ends by running **`plstatus`** to confirm what is running.

While the CP is ready to receive datasets to process it is usual to run the blackboard monitor **`simplemon`** . It is "simple" because it only depends on a terminal window that is cleared and redrawn as the blackboard changes. So you should open a dedicated terminal window for monitoring, login to the operator host, and run the monitor there.

The monitor is far from simple it what it can monitor (see the help). Because querying all pipelines on all nodes can be slow you should use one of the aliases **`sm`** or **`smfil`**.

Finally, the opposite of **`plrun`** is **`plstop`** . This stops everything on all the nodes. As with **`plrun`** the stop command ends with **`plstatus`** to confirm the status of the pipeline. To stop the simple monitor just requires cntrl-C in the window.

Stopping the pipeline is normally only done when things get in a bad state, for an emergency kill of datasets, or to gracefully prepare for a cluster shutdown.

Figure 3: Output of **plstatus** when the pipeline is not running. Host dec02 is down. In this example a subshell is used to temporarily change **PROCNODES** to shortten the output.

```
deccp@dec01$ (setenv PROCNODES dec01,dec02; plstatus)
Pipeline node status report: Tue Jan  7 14:18:03 2020
Current node is dec01

Node        Connected   NodeMgr     Available Pipelines
-----------------------------------------------------------------
dec01        Yes          Down
-----------------------------------------------------------------
dec02        No           Down
-----------------------------------------------------------------
```

Figure 4: Output of **plstatus** for running pipeline. In this example a subshell is used to temporarily change **PROCNODES** to shortten the output.

```
deccp@dec01$ (setenv PROCNODES dec01,dec02; plstatus)

Pipeline node status report: Tue Jan  7 14:12:57 2020
Current node is dec01

Node        Connected   NodeMgr     Available Pipelines
-----------------------------------------------------------------
dec01        Yes          Running      src:
                                       ccd, cor, dts, fil, fsi,
                                       ill, nsa, omi, rsp, ssk,
                                       sss, stk, top, xtk, zsi
-----------------------------------------------------------------
dec02        Yes          Running      src:
                                       ccd, cor, fsi, ill, nsa,
                                       omi, rsp, ssk, sss, stk,
                                       xtk, zsi
-----------------------------------------------------------------
```

Figure 5: Starting the pipeline from scratch with **plrun** .

```
deccp@dec01$ plrun
=====================================================================
Starting the pipeline application DCP
Tue Jan  7 15:03:11 MST 2020
Data manager node = dec01
Directory server node = dec01
Submit nodes = dec01
Processing nodes = dec01,dec02
Cleaning pipeline local working directories ...
Starting the name server...
Starting the directory server...
Starting the node managers...
Starting the pipelines on the data manager node...
Starting the pipelines on the processing nodes...
```

Followed by the output of **plstatus**

7 Triggering Datasets

Processing a dataset consists of triggering the pipeline with a list of exposures in the archive. The highest level command to trigger a dataset is **topnext** . This depends on a scheduling file that calls **plpsq** . We will return to **topnext** at the end of this section and start with **plpsq** .

Besides the primary purpose of creating the list of exposures and triggering the pipeline, **plpsq** defines the dataset name which appears throughout the processing and the final data products directory. There are many options with this command which are useful for the operator (see the help).

While the infrastructure has few requirements on the dataset names, the operator should follow the CSDC convention which has the form

[PLQUEUE]_[PLQNAME]_[PLPROCID].

These components have a useful purpose in structuring the final data products files (though the archived data product file names are changed to follow the archive naming conventions**?**). They are also recorded in the data products headers.

PLQUEUE is, by convention, DECxxx, where xxx is the semenster such as 19B. Occasionally other queue names might be used such as for target of opportunity (e . g. TOO19B) or special programs (e .g . DES14B). The queue name is also significant for automatically triggering the next dataset from a predefined operator queue file. **PLQNAME** , by convention, has a prefix and the first calendar date of the data (e .g . AZ20200103). **PLPROCID** is a processing identification that allows dataset names to be unique. It is set by **plpsq** and is a packed version of the date and time the dataset was triggered (see also **date2procid** and **procid2date**). If needed the **PLPROCID** field can be overridden with options.

A basic example of the **plpsq** command is

```
plpsq DEC19B_AZ 20200103 20200105
```

The first argument is the queue with the name prefix to which the second argument is appended. The data to be triggered is found from the list of calendar dates.

The list of files selected by **plpsq** is purely set by the observing calendar dates. However, it does have options (**-c** and **-o**) to only select calibrations exposures or only science exposures. Finer selections are done by the pipeline during staging and grouping. **plpsq** passes on requests on to the pipeline (in comments at the bottom of the trigger lists) with its options. The most used are for only certain filters or proposal IDs.

It is common for two (or more) programs to share an observing night and for runs (multiple nights of the same program) to be interleaved in a complex way. While it is possible to process data in blocks without regard to the programs, the convention is to process programs separately as their runs end. The fundamental reason for this is that different programs may require different processing parameters.

Defining datasets by program is controlled by the proposal IDs. As noted earler, a request to only include certain proposal IDs in a dataset can bey done with an option to **plpsq** . It can also be done with operator configuration file described in section **??**.

7.1 topnext

An important feature of the pipeline system is defining a queue of datasets in advance. There can be different types of queues but in standard operation a queue consists of the datasets from a semester, particularly the current one. It is possible to set up a queue from the published telescope schedule.

A queue file is set up in the operator directory reference by any of **$HOME/Operator**, **$Operator**, or **$PipeConfig**. The filename is **<queue>next.dat** ; e .g . DEC19Bnext.dat. The file consists of a list of **plpsq** commands. Blank lines and hash comments are ignored. When **topnext <queue>** is executed the first unignored command is executed and that line is commented. The special command **---** is a no-op placeholder which means when it is the next statement to be executed it will be commented but nothing will be done. This is one way to have the automatic chaining of datasets stop for the completing dataset.

A new queue file is created by the operator with an editor in the Operator runtime directory. Subsequently, the convenience alias **plqueue <queue>** can be used to edit (with **vi**) without dealing with the directory path or `next.dat` in the filename.

Allowing the pipeline to trigger a new dataset when the current processing completes is a very useful operator feature. This is done by having the last step of the pipeline execute the command **topnext** with the queue name of the dataset being finished.

Figure 6 shows an example of a piece of a queue file for queue **DEC19**. Each dataset has a stylistic operator comment. The first **plpsq** (Zenteno) was commented automatically as it was processed by **topnext** . The next dataset, Shen, will be executed either when the pipeline finishes or the operator runs **tonext**. The example also shows use of **---** which will be a no-op execution.

There are two other ways the operator may control the chaining of datasets by the pipeline. One is to use the **-n** flag in **plpsq** and the other is to change the blackboard to mark the ***topnext*** stage as completed (see §**??** on manipulating the blackboard).

In summary, the three typical ways to trigger a dataset for processing are **plpsq** and **topnext** from the command llne and **topnext** executed by the pipeline at the end of a dataset. To avoid a new dataset being triggered add a **---** to the queue file or change the blackboard for the ***topnext*** stage.

Figure 6: Example queue file DEC19Bnext.dat.

```
[...]

# Zenteno 2019B-0323
# plpsq -p 323 DEC19B_AZ 20200105

# Shen 2019B-0910
plpsq -p 910 DEC19B_SY 20200120

# Rest 2018B-0122
plpsq -p 122 DEC19B_AR 20200121

# Drlica-Wagner 2019A-0305
plpsq -p 305 -c DEC19B_AD 20200120 20200121

---

# Dawson 2018A-0273
plpsq -p 273 DEC19B_WD 20200122 20200123 20200124

[...]
```

8 The Blackboard

The processing state of the Community Pipeline is kept by the NHPPS *blackboard*. It records what datasets and stages are processing, completed, pending, or stopped for some reason. It also records the count of child pipelines farmed out by a parent pipeline. And not only does it show the state but it also controls the workflow since trigger conditions for a stage to execute are generally based on the status flag of other stages. This means that by modifying the blackboard the operator exerts control over the CP.

In a complex pipeline application distributed over many nodes and processing many pieces of the data in parallel the blackboard can be quite large. This means constantly monitoring the entire blackboard is not only impractical but slow due to the need to query all the pipeline managers on all the nodes for their pieces of the blackboard. Instead the operator normally monitors the top level parent pipelines, which run only on the data manager node, and the counts of data scattered to the child pipelines distributed across the nodes. As long as the counts are steadily decreasing the pipeline is operating without problems. Only when there are problems does the operator drill down through the blackboard.

One aspect of the blackboard that hasn't been explicitly stated is that, by convention, the last stage of a pipeline erases its blackboard entry provided none of the flags show an unusal condition. This erasing of entries is important to keep the blackboard from growing indefinitely and making monitoring nearly impossible. It is stated here because if the operator needs to determine if a dataset has completed other methods than the blackboard must be used.

There are a number of operator commands which interact with the blackboard. As mentioned earlier, operation of the CP has been made convenient by structuring low level commands within a few higher level commands. These higher level operator command are discussed below. You will notice that most of these have names beginning with **`osf`** which stands for *operation status flag*. While each stage has a status flag, represented by a character, the set of dataset flags in a pipeline entry can be displayed and interacted with as a string. Therefore operator tools for the blackboard can operate on an subset or entire entry as a single unit rather than having to address each flag separately.

8.1 Status Flags

The NHPPS blackboard status flags are recorded by a single character. It is up to the pipeline application to define the characters and meanings for the flags. To be meaningful to the operator they need to be mnemonic and consistent. The CP uses the following consistent conventions.

When a dataset is created the flag for the first *start stage*, **`___start`** , is **`p`** and all other stages are **`_`** . The mnemonics for these are *processing* and *waiting to execute*. Techically the start stage is not processing but it indicates the dataset is processing and the flag will stay **`p`** until the dataset finishes. The use of underscore is to make blackboard displays appear like an execution status is pending though technically it is just a status universally used as one of the trigger conditions for a stage to execute. If the stage's status flag is not underscore then it cannot execute. Therefore, this is one of the ways an operator can control execution through the underscore flag.

Table 3 tabulates the conventional flags seen in the CP. The most common flags beyond underscore are **p**rocessing and **c**ompleted normally.

The **w**ait flag is used only in stages where the pipeline is waiting for the count of child datasets to decrement to zero. In almost all cases the scatter/gather part of a parent pipeline consists of three stages. The first builds the trigger lists and has a stage name like ___**2**___ , where the underscores are the three character parent and child pipelines. The second sends some data to the child pipelines and keeps some to trigger when a child dataset returns. It has names like ___**C**___. The last is a stage that receives the return triggers from the child pipeline, decrements the count, and sends any new pending dataset to the node that returned data. It has names like ___**R**___ and its status flag is **w** until the count decrements to zero when it is set to completed so the next stage will trigger. Understand this logic is useful to the operator in some control situations.

Ideally, a dataset will progress through the pipeline stages until all stages except the first and last are **c** . The **d** flag is for the ___**done** stages and are not usually seen since immediately after completion the dataset is erased from the blackboard. The no data flag appears at times and may or may not be a blocking condition. The halt flag is used to pause processing either for developer diagnostics or to temporarily pause some datasets for operational reasons such a running out of disk space or letting other datasets make progress without losing work already done.

Clearly the error and fatal error flags are the ones that require operator attention. The **e** is rarer and generally signals a type of error that can be trapped and acted upon. The **f** error occurs when a stage executable exits unexpected. One case when this happens over all pipelines is when there is no disk space. Another is due to a crash. Sometimes the operator can recover from this state for a particular dataset by retrying the stage.

Every stage has an maximum execution time. When a stage command does not complete in that time the **x** flag is raised. The command may or may not be terminated. In the latter case the operator has to find and kill the command.

The **t** flag is rare and should not occur if the pipeline startup was successful.

8.2 osfbb

Typically the view selected for **simplemon** and its aliases is just top level pipelines. This is fine for general monitoring. But when you want to drill down or do a quick check from shell window or a special script or alias the command **osfbb** is used. This is one of the more frequently used commands in the operator's toolkit.

With its flags to select the kind of blackboard dump desired it has several common uses. With only the required argument of a pipeline (or list of pipelines) a dump of the current datasets and their flags is output.

Table 3: Table of CP status flag convention.

_	=	waiting to execute
p	=	processing/running
c	=	completed normally
w	=	waiting for child returns
n	=	no data / no processing
d	=	dataset done
e	=	error (sometimes not blocking)
f	=	fatal error
h	=	halt
s	=	skip
t	=	a child pipeline cannot be found
x	=	timeout - stage exceed expect runtime
W	=	waiting for operator response
N	=	operator response of no
Y	=	operator response of yes

8.3 osffix

8.4 osfwait

8.5 osfrm

8.6 osf_ckpoint

8.7 simplemon

8.8 osfbb

A Pipelines and Stages

A.1 top

The **top** pipeline operates on a full path list of unprocessed DECam exposures and produces calibrated data products in the final pipeline data products directory. The list can include both calibration and science exposures and all filters which the pipeline will orchestrate. This is the starting point of the Community Pipeline triggered by the **plpsq** command. It runs only on the data manager node with up to four instances of each stage.

topstart The standard start stage that creates the dataset blackboard entry, creates the dataset working directory, and puts the input trigger file in working directory.

topblock A special purpose stage that limits the number datasets running at the same time to the number of instances of this stage that can run simultaneously. It is used in circumstances where datasets are triggered automatically. Its only use was to process blocks of exposures in near real-time triggered by the observer during the night. It is controlled by the environment variable **OSF_BLOCK**.

topstage Copy the input raw exposures from the archive to the working directory. Exclude files in the operator ignore lists. Only stage exposures with proposal IDs requested by the **objprops** and **calprops** parameters or in the **plpsq** flag. Only stage object or calibration exposures if requested in the **plpsq** flag. If the stage only option was selected in **plpsq** then halt.

topgroup Exclude exposures with certain bad data conditions: missing/invalid keywords, missing extensions, and substrate voltages off. Exclude exposures with titles containing the words in table 2). Group the exposures by type – zero/bias, dome flat, and science. The zero and dome flat exposures are further grouped by filter and night and if a master calibration for a night already exists in the calibration library they are skipped. Do additional grouping on the science exposures, as requested by the **topgroup** parameter, by filter (the default), by filter+night, or by filter+sequence. Note it is possible to not group by filter and lower level pipelines will handle the filter. The purpose of groupings other than just by filter are to have additional control over stacking. Also note the operator has control over grouping by the way datasets are defined; in particular, in the **topnext** queue definitions (by night) and with **plpsq** and explicit exposure lists.

topCzmi Trigger the **zmi** pipeline with any nightly zero datasets needing to be done.

topRzmi Wait for each nightly master zero calibration to be completed by the **zmi** pipeline.

topCfmi Trigger the **fmi** pipeline with any nightly dome flat datasets needing to be done.

topRfmi Wait for each nightly master dome flat calibration to be completed by the **fmi** pipeline.

topCfil Trigger the **fil** pipeline with the science groupings made by **topgroup** .

topRfil Wait for each science dataset grouping to be completed by the **fil** pipeline.

top2sta If requested by the **stk_sta** parameter create stack-of-stacks datasets from the stacks produced by the **fil** datasets.

topCsta Trigger the **sta** pipeline to make stacks-of-stacks.

topRsta Wait for each stack-of-stacks data products to be completed by the **sta** pipeline.

top2pft Create the photometric flat calibration input dataset for the **pft** pipeline.

topCpft Trigger the **pft** pipeline to make the photometric flat calibrations.

topRpft Wait for the photometric flat calibrations to be completed by the **pft** pipeline.

topsave Save the log files from the **top** pipeline working directory to the permanent pipeline repository. Delete the staged data files. Remove all the working directories for the child pipelines of this dataset.

topnext Call **topnext** with the same queue name of the completed dataset. The operator may set the status of this stage to **c** in advance to triggering a new dataset since the condition to finish requires **topsave** to also be completed.

topdone The standard done stage which includes deleting any temporary working files from the dataset directory and erasing the dataset from the blackboard if the status flags are all as expected.

A.2 fil

The **fil** pipeline operates on a list of unprocessed DECam science exposures and produces calibrated data products in the final pipeline data products directory. It is a child pipeline of the **top** pipeline. In most cases the input is from a single filter, hence the mnemonic. The data products are single exposures that are unresampled and projected, with two levels of sky subtraction, and stacks. This pipeline is largely and orchestrator of child pipelines so it runs only on the data manager node with up to for instances of each stage.

filstart The standard start stage that creates the dataset blackboard entry, creates the dataset working directory, and puts the input trigger file in working directory.

fil2xtk Create datasets for the **xtk** pipeline. The datasets are for each exposure using pipeline standard names. The child pipeline separates the multiextension file into individual CCDs while applying crosstalk, overscan, and header repair operations.

filCxtk Standard call stage for the **xtk** pipeline.

filRxtk Wait for all datasets (exposures) to be completed by the **xtk** pipeline.

fil2ccd Create datasets for the **ccd** pipeline. The datasets are groups of all images from the same ccd. The child pipeline returns images with standard dome calibrations.

filCccd Standard call stage for the **ccd** pipeline.

filRccd Wait for all datasets to be completed by the **ccd** pipeline.

fil2omi Create datasets for the **omi** pipeline. The datasets are all ccd files from a single exposure. For disk space management all the bulk files produced by the **xtk** pipeline are deleted.

filComi Standard call stage for the **omi** pipeline.

filRomi Wait for all datasets (exposures) to be completed by the **omi** pipeline.

fil2rsp Create datasets for the **rsp** pipeline which have a successful astrometric calibration. The datasets are all ccd files from a single exposure. For disk space management all the bulk files produced by the **ccd** pipeline are deleted. This stage determines the projection tangent point for each exposure. In almost all cases the tangent points are from a healpix grid on the sky so that nearby exposures have the same reprojection to allow stacking. One or two versions of each ccd are returned. These are with and without sub-ccd scale sky subtraction.

filCrsp Standard call stage for the **rsp** pipeline.

filRrsp Wait for all datasets to be completed by the **rsp** pipeline.

fil2stk Create datasets for stacking and (possibly) moving transient detection by the **stk** pipeline. The grouping is by common tangent point resampling of two or more. If the number of exposures in a dataset is larger than **stk_maxstack** then multiple datasets are defined.

If moving transient detection is selected by **stk_mov** > 0 and the number of exposures is larger than **stk_maxseq** then multiple datasets are created in addition to the dataset for all exposures. In this case the moving transient detection datasets do not produce stacks.

filCstk Standard call stage for the **stk** pipeline.

filRstk Wait for all datasets to be completed by the **stk** pipeline.

filmov If transient detections were made this produces some additional data products for the detections.

fil2nsa Create datasets for producing the single exposure archive data products from the **omi** and **rsp** results.

filCnsa Standard call stage for the **nsa** pipeline.

filRnsa Wait for all datasets (exposures) to be completed by the **nsa** pipeline.

fildp Create the operator review html page.

fildone The standard done stage which includes deleting any temporary working files from the dataset directory and erasing the dataset from the blackboard if the status flags are all as expected. Instead of directly returning to the parent pipeline (**top**) it triggers the **dts** pipeline which then returns to the parent pipeline.

A.3 zmi

The **zmi** pipeline operates on a list of unprocessed DECam zero/bias exposures from a night and produces master calibrations for the CP and as data products in the final pipeline data products directory. It is a child pipeline of the **top** pipeline. As noted elsewhere, **topgroup** automatically excludes zero datasets for nights where a master zero has already been produced and added to the CP calibration library which means the **zmi** pipeline is skipped for those nights.

zmistart The standard start stage that creates the dataset blackboard entry, creates the dataset working directory, and puts the input trigger file in working directory.

zmi2xtk Create datasets for the **xtk** pipeline. The datasets are for each exposure using pipeline standard names. The child pipeline separates the multiextension file into individual CCDs while applying crosstalk, overscan, and header repair operations.

zmiCxtk Standard call stage for the **xtk** pipeline.

zmiRxtk Wait for all datasets (exposures) to be completed by the **xtk** pipeline.

zmi2zsi Create datasets for the **zsi** pipeline. The datasets are groups of all images from the same ccd. The child pipeline applies standard ccd calibrations and returns a master (combined) zero calibration.

zmiCzsi Standard call stage for the **zsi** pipeline.

zmiRzsi Wait for all datasets to be completed by the **zsi** pipeline.

zmi2nsa Create the standard data products and review graphics for the archive.

zmireview If the **zmi_review** parameter is no (the default) then the stage simply exits with **c**ompleted and pipeline goes on. If the parameter is yes, however, then the stage status flag is set to operator wait, **W**, which blocks the pipeline from going on to the next stage. The operator can then review the html graphics page and decide whether to accept the calibration for the calibration library. To continue the operator modifies the stage blackboard entry to **Y**es or **N**o. The former causes the stage to run again and exit with **c**ompleted and the latter to exit with **n**o data. If the stage flag is **n** the **zmiputcal** stage is skipped (not triggered) and its status flag is set to **c**ompleted. In either case the operator response is logged in **$Operator/opwait.log**. If, for some reason, the same calibration is encountered again the recorded response is used rather than generating an operator wait status.

zmiputcal The master calibration is put into the calibration library. Note that the **zmireview** stage and subsequent operator interaction may lead to this stage not being triggered.

zmidone The standard done stage which includes deleting any temporary working files from the dataset directory and erasing the dataset from the blackboard if the status flags are all as expected. Instead of directly returning to the parent pipeline (**top**) it triggers the **dts** pipeline which then returns to the parent pipeline.

A.4 fmi

The **fmi** pipeline operates on a list of unprocessed DECam dome flat exposures from a night and filter and produces master calibrations for the CP and as data products in the final pipeline data products directory. It is a child pipeline of the **top** pipeline. As noted elsewhere, **topgroup** automatically excludes dome flat datasets for nights where a master flat has already been produced and added to the CP calibration library which means the **fmi** pipeline is skipped for those nights.

fmistart The standard start stage that creates the dataset blackboard entry, creates the dataset working directory, and puts the input trigger file in working directory.

fmi2xtk Create datasets for the **xtk** pipeline. The datasets are for each exposure using pipeline standard names. The child pipeline separates the multiextension file into individual CCDs while applying crosstalk, overscan, and header repair operations.

fmiCxtk Standard call stage for the **xtk** pipeline.

fmiRxtk Wait for all datasets (exposures) to be completed by the **xtk** pipeline.

fmi2fsi Create datasets for the **fsi** pipeline. The datasets are groups of all images from the same ccd. The child pipeline applies standard ccd calibrations and returns a master (combined) dome flat calibration.

fmiCfsi Standard call stage for the **fsi** pipeline.

fmiRfsi Wait for all datasets to be completed by the **fsi** pipeline.

fmi2nsa Create the standard data products and review graphics for the archive.

fmireview If the **fmi_review** parameter is no (the default) then the stage simply exits with **c**ompleted and pipeline goes on. If the parameter is yes, however, then the stage status flag is set to operator wait, **W**, which blocks the pipeline from going on to the next stage. The operator can then review the html graphics page and decide whether to accept the calibration for the calibration library. To continue the operator modifies the stage blackboard entry to **Y**es or **N**o. The former causes the stage to run again and exit with **c**ompleted and the latter to exit with **n**o data. If the stage flag is **n** the **fmiputcal** stage is skipped (not triggered) and its status flag is set to **c**ompleted. In either case the operator response is logged in **$Operator/opwait.log**. If, for some reason, the same calibration is encountered again the recorded response is used rather than generating an operator wait status.

fmiputcal The master calibration is put into the calibration library. Note that the **fmireview** stage and subsequent operator interaction may lead to this stage not being triggered.

fmidone The standard done stage which includes deleting any temporary working files from the dataset directory and erasing the dataset from the blackboard if the status flags are all as expected. Instead of directly returning to the parent pipeline (**top**) it triggers the **dts** pipeline which then returns to the parent pipeline.