CRDS-to-DMS Instrument Reference File Delivery Interface

The Calibration Reference Data System (CRDS) has been designed and implemented for HST, replacing the database-centric model of Calibration Data Base System (CDBS) with explicit reference file rule definitions in text files (using Python dictionary format). CRDS is being implemented for JWST, in DMS build 5. CRDS is used to deliver instrument reference files, along with CRDS mapping files, to DMS via a catalog “manifest” file.

The software that receives and processes the catalog files from CRDS has two distinct parts, CRDS File Poller and CRDS File Delivery (the OWL/Condor workflow), and is facilitated by the OWL Shoveler. Metrics data about each delivery are stored in the JWST Data Processing Metrics database tables *CrdsCatalog* and *CrdsDataFile*.

# CRDS Deliveries

CRDS delivers catalog (.cat) files, mapping files (\*.?map), and instrument reference files (\*.fits). Each catalog file identifies a set of mapping and/or reference files that constitute a delivery. *Note that the font colors used in the subsections below are an attempt to provide mapping of a particular file throughout the subsections.*

## Catalog File

Catalog files are manifests that clearly identify the contents of a CRDS delivery.

* Filename: jwst\_<delivery number>.cat
* 1 per delivery; required
* catalog files are simply delivery mechanisms and are not archived
* example - jwst\_21.cat: ***NEED a JWST example here; don’t have one***

hst\_0017.pmap

hst\_acs\_0004.imap

hst\_acs\_ccdtab\_0001.rmap

hst\_acs\_pctetab\_0001.rmap

xa81715gj\_ccd.fits

xa81724cj\_cte.fits

## Pipeline Mapping File

This file defines a CRDS “pipeline context”; it identifies the instrument mapping file versions (.imap) that describe that particular context.

* <mission>\_<context number>\*.pmap
* 0 or 1 per catalog/delivery:
  + a catalog file can sometimes deliver only reference (.fits) files (i.e., no mapping files)
  + if a .pmap file is included, there is only one
* pipeline mapping files are archived as Calibration Reference Files (data\_level CalRef)
* example - hst\_0017.pmap: ***NEED a JWST example here; don’t have one***

header = {

'derived\_from' : 'hst\_0016.pmap',

'description' : 'Initially generated on 2011-11-16 10:23:00',

'mapping' : 'PIPELINE',

'name' : 'hst\_0017.pmap',

'observatory' : 'HST',

'parkey' : ('INSTRUME',),

'sha1sum' : '93406a5a8fd49b981047ad459483f1db9a459176',

}

selector = {

'ACS' : 'hst\_acs\_0004.imap',

'COS' : 'hst\_cos\_0003.imap',

'NICMOS' : 'hst\_nicmos.imap',

'STIS' : 'hst\_stis\_0005.imap',

'WFC3' : 'hst\_wfc3\_0005.imap',

'WFPC2' : 'hst\_wfpc2.imap',

}

## Instrument Mapping Files

Each of these files defines a CRDS “instrument context”; it identifies the reference type mapping file versions (.rmap) that define that instrument’s reference file selection rules.

* <mission>\_<instrument>\_<sequence number>.imap
* 0 – 6 per catalog/delivery:
  + 0 or 1 per instrument (***acs, cos, nicmos, stis, wfc3, wfpc2***) ***NEED JWST instruments***
  + a catalog file can sometimes deliver only reference (.fits) files (i.e., no mapping files)
  + if the catalog includes a pmap file, there is also at least one imap file
  + there is never more than one per instrument
* instrument mapping files are archived as Calibration Reference Files (data\_level CalRef)
* example - hst\_acs\_0004.imap: ***NEED a JWST example here; don’t have one***

header = {

'derived\_from' : 'hst\_acs\_0003.imap',

'instrument' : 'ACS',

'mapping' : 'INSTRUMENT',

'name' : 'hst\_acs\_0004.imap',

'observatory' : 'HST',

'parkey' : ('REFTYPE',),

'sha1sum' : '30cdfdd324e6db25ac8698d78fd2d50d1ca46cf3',

}

selector = {

'atodtab' : 'hst\_acs\_atodtab.rmap',

'biasfile' : 'hst\_acs\_biasfile\_0002.rmap',

'bpixtab' : 'hst\_acs\_bpixtab.rmap',

'ccdtab' : 'hst\_acs\_ccdtab\_0001.rmap',

'cfltfile' : 'hst\_acs\_cfltfile.rmap',

'crrejtab' : 'hst\_acs\_crrejtab.rmap',

'd2imfile' : 'hst\_acs\_d2imfile\_0001.rmap',

'darkfile' : 'hst\_acs\_darkfile\_0002.rmap',

'dgeofile' : 'hst\_acs\_dgeofile.rmap',

'drkcfile' : 'hst\_acs\_drkcfile\_0002.rmap',

'flshfile' : 'hst\_acs\_flshfile.rmap',

'idctab' : 'hst\_acs\_idctab.rmap',

'imphttab' : 'hst\_acs\_imphttab.rmap',

'mdriztab' : 'hst\_acs\_mdriztab.rmap',

'mlintab' : 'hst\_acs\_mlintab.rmap',

'npolfile' : 'hst\_acs\_npolfile.rmap',

'oscntab' : 'hst\_acs\_oscntab.rmap',

'pctetab' : 'hst\_acs\_pctetab\_0001.rmap',

'pfltfile' : 'hst\_acs\_pfltfile.rmap',

'shadfile' : 'hst\_acs\_shadfile.rmap',

'spottab' : 'hst\_acs\_spottab.rmap',

}

## Reference Type Mapping Files

Each of these files defines a CRDS reference file mapping rule set for a given instrument and reference file type. It identifies the active reference files for that instrument and reference file type, along with the file selection criteria used to map a science exposure to a particular active reference file.

* <mission>\_<instrument>\_<reference file type>\_<sequence number>.rmap
* 0 - n per catalog/delivery; 1- n per imap file
* reference type mapping files are archived as Calibration Reference Files (data\_level CalRef)
* example - hst\_acs\_pctetab\_0001.rmap: ***NEED a JWST example here; don’t have one***

header = {

'derived\_from' : 'hst\_acs\_pctetab.rmap',

'filekind' : 'PCTETAB',

'instrument' : 'ACS',

'mapping' : 'REFERENCE',

'name' : 'hst\_acs\_pctetab\_0001.rmap',

'observatory' : 'HST',

'parkey' : (('DETECTOR',), ('DATE-OBS', 'TIME-OBS')),

'reffile\_format' : 'TABLE',

'reffile\_required' : 'NO',

'reffile\_switch' : 'PCTECORR',

'rmap\_relevance' : '((DETECTOR == "WFC") and (PCTECORR != "OMIT"))',

'sha1sum' : '9e4c4e99f020249b56f1cc5b70fb36ee6abf2b9e',

}

selector = Match({

('WFC',) : UseAfter({

'2002-03-01 00:00:00' : 'xa81724cj\_cte.fits',

}),

})

## Instrument Reference Files

The instrument reference files are created by the instrument teams. They are used by other CRDS tools, bestref software, and calibration software.

* <unique coded timestamp><instrument character>\_<ref type>.fits ***Is this correct??***
* The instrument character is one of ***NEED JWST instruments***
  + ***j = acs***
  + ***l = cos***
  + ***n = nicmos***
  + ***o = stis***
  + ***i = wfc3***
* 0 - n per catalog/delivery; a catalog file can sometimes deliver only mapping files (i.e., no .fits files)
* files are binary FITS file format, and can be images, tables, or combinations
* all CRDS reference files are archived as Calibration Reference Files (data\_level CalRef)
* An example filename is xa81724cj\_cte.fits. The contents are not included here, as the file is binary.

# CRDS File Poller

The poller is stand-alone, continuously-running software that monitors the disk area mutually agreed upon by CRDS and DMS for CRDS to place deliveries.

## Configuration Files

Poller configuration is provided by standard DMS .resource/.path files.

The .resource file is delivered with the software build, in the following location. The values in this file are resolved at run time via the user’s sdp.path file.

$PIPELINE\_BASE/defs/poller\_crdsRef.resource

Each user has their own sdp.path file, generated from a template after a DMS software delivery, which is by default accessed from the user’s home directory (/home/<user>/overrides/defs/):

$OPUS\_DEFINITIONS\_DIR/sdp.path

## Start the CRDS Poller

Currently, the CRDS poller must be started by the operator using the following command\*:

* + - start\_poller.csh CRDS

\*NOTE: In a future release, this (and all OWL file pollers) may be started from the OWL GUI.

Once the poller is started, it will run until it is intentionally stopped by the operator (or it crashes).

The script will not start a new CRDS poller if one is already running.

## Check the CRDS Poller

To see if a CRDS poller is running:

* + - got\_poller.csh CRDS

Like the Shoveler, if no running poller is found, there is no output from the script. If a poller is running, output similar to the following example is displayed:

lthomp 9392 1 1 14:16 pts/9 00:00:00 python /usr/stsci/pipeline/builds/jwstdp/JWSTDP-2015\_1-150326/bin/manifest\_file\_poller.py CRDS

## Stop the CRDS Poller

To stop a running CRDS poller:

* + - stop\_poller.csh CRDS

File pollers stop themselves when they find a shutdown file. The content of the file is not important; it is not opened or read. Just the existence of the file causes the poller to terminate. This script reads the configuration files and creates the proper shutdown file in the proper location.

The frequency at which the poller checks for the shutdown file is specified in the OWL database, PollerHeartbeat.intervalSeconds. The default value is 60 seconds, but may be changed. It should not be set to a long time, because if the system needs to come down, the poller needs to respond to the shutdown order quickly.

## CRDS Poller Log Files

Each time the CRDS poller is started it opens two new log files. When it is stopped the log files are closed. Log files are in the $PIPELINE\_LOG\_DIR directory.

The main log file is named CRDS\_Poller\_YYYYMMDD\_hhmmss.log, where the time stamp reflects when the poller was started.

An additional log file is called start\_CRDS\_file\_poller.log and is overwritten at each CRDS poller start. Unless the poller fails to start or crashes, this log file should be empty; otherwise error messages are captured in this file.

## CRDS Poller Processing

### Look for a new catalog file

The CRDS poller monitors a directory, watching for a catalog file to appear:

* The directory to be monitored can exist on the same host where the poller is running (local poller) or on a remote host (remote poller). The configuration files allow either type of poller to be defined via the REM\_HOST and REM\_USER keywords.
* The delivery directory to be monitored is specified in the configuration files as keyword SRC\_DIR.
* The FILE\_MASK keyword in the configuration files tells the CRDS poller how to identify a catalog file. Its JWST template value is *jwst\_[0-9]\*.cat* and this value must not be changed.
* The frequency at which the poller checks for a new catalog file is specified as keyword MANIFEST\_WAIT\_SECS. The default value is 1800 seconds (30 minutes). This value may be changed, but should not be significantly shortened. It should be long enough for all the files delivered by a catalog file to be archived before processing of another catalog file begins. This is to guarantee that the deliveries are processed in the order in which CRDS delivers them, which is important because files can build on previously-delivered files.

### Submit the new catalog file for processing

When the CRDS File Poller finds a new catalog file, it does the following:

* copies the catalog file to the workflow processing area (keyword DEST\_DIR in the configuration files)
* inserts a crds\_delivery workflow request in the DPQ (identified by the DPQ\_DB and DPQ\_SERVER keywords)
* renames the original catalog file in the delivery directory from \*.cat to \*.cat\_submit, to indicate it has been submitted for processing

# CRDS File Delivery

Nominally, the operator has no interaction with the CRDS File Delivery software, which processes an entire catalog file/delivery. The processing is initiated by the Shoveler in response to the poller-inserted ‘crds\_delivery’ workflow request:

* the Shoveler, based on the ‘crds\_delivery’ *workflowType*, executes the crds\_delivery.py plugin, passing it the DPQ request entry
* crds\_delivery.py identifies the crds\_delivery.dag template for the work, and executes process\_dag.py, passing it the data it needs
* process\_dag.py renders the templates into crds\_delivery\_<catalogFileRoot>.dag, and the DAG-specified job files, crds\_submit\_<catalogFileRoot>.job and crds\_confirm\_<catalogFileRoot>.job, submits the .dag file to HTCondor, and captures the HTCondor-assigned workflow ID

## Configuration Files

CRDS File Delivery configuration is provided by standard DMS .resource/.path files.

The.resourc file is delivered with the software build, in the following location. The values in this file are resolved at run time via the user’s sdp.path file.

$PIPELINE\_BASE/defs/crds\_delivery.resource

Each user has their own sdp.path file, generated from a template after a DMS software delivery, which is by default accessed from the user’s home directory (/home/<user>/overrides/defs/):

$OPUS\_DEFINITIONS\_DIR/sdp.path

## CRDS Workflow Processing Directories

The processing of a CRDS catalog file can create many new directories in the $OWL\_LOGS area. For example, if a CRDS catalog file delivers 25 data files, 26 new processing directories are created: one directory for the processing of the catalog file and one directory for the ingest of each data file. Some deliveries are comprised of only a few files, but some can be very large.

The naming convention for the catalog processing directory is: user\_”jwst”\_deliveryNumber\_elapsedTime. Using sample catalog name jwst\_5.cat as an example (deliveryNumber = 5), a directory might be lthomp\_jwst\_5 \_1397034279.006865 and the log files an operator cares about in this directory are:

ALOG\_1397034284\_crds\_submit\_jwst\_5.err

ALOG\_1397034284\_crds\_submit\_jwst\_5.out

ALOG\_1397034288\_crds\_confirm\_jwst\_5.err

ALOG\_1397034288\_crds\_confirm\_jwst\_5.out

For CRDS, if the workflow completed, both .err files are empty.

The submit job actually creates an ingest request for each data file and HTCondor/OWL creates a separate workflow processing directory for each request, named like: user\_datafileRoot\_elapsedTime. Using n1a00001a.fits as an example, a directory might be lthomp\_n1a00001a\_1397033708.765211 and the log files of interest are:

ALOG\_1397033717\_RQ\_ArchiveSubmit\_n1a00001a.err

ALOG\_1397033717\_RQ\_ArchiveSubmit\_n1a00001a.out

Because of the way ingest\_crds.py, and the software it calls, is written, regular log messages are captured in the .err file. The .out file contains only some of the database queries that are executed.

## crds\_submit Job

The goals of the submit step are to verify that all data files in the catalog file are available, get copies of them into the workflow processing area, and submit archive ingest requests for each of them.

### Processing Tasks

* rename the original catalog file in the delivery area from .cat\_submit to .cat\_proc
* read the local catalog file and identify all the data files
* copy the data files to the workflow processing area (CRDS\_ARCH\_DIR in sdp.path)
* insert an ‘ingest\_CalRef’ workflow request for each data file into the *DpQueue* table
* create a $DATA\_ROOT\_DIR/crds/temp/<catalog fileset>.pkl file to hold the processing information that will be needed by the confirm step

The Shoveler finds the new requests and kicks off [ingest](#ingest_owl) for each data file.

### Source Failures

CRDS catalog deliveries are handled as all-or-nothing; if the catalog file cannot be read or any of the data files cannot be found in the SRC\_DIR (i.e., the problem is in the CRDS delivery area) the following occurs:

* rename the catalog file in the delivery area from .cat\_proc to .cat\_ERROR
* remove files from the workflow processing area
* send a failure email to the email addresses identified by the EMAIL\_LIST keyword
* terminate processing

The CRDS team will have to fix the problem and redeliver a catalog file.

## crds\_confirm Job

The goals of the confirm step are to verify that all the data files are archived, record metrics about the delivery, clean up, and send an email announcing the CRDS delivery is complete.

### Processing Tasks

* read the temporary .pkl file to obtain information about the data files that were submitted to ingest
* query the DADS archiveFile service in a loop until all files are determined to be archived or a timeout occurs; the process times out when it has queried the maximum number of times (MAX\_QUERIES keyword, default value = 10) and one or more files is not archived; the frequency of the queries depends on the value of keyword QUERY\_REST\_PERIOD (default is 300 seconds, or 5 minutes)
* when all files are determined to be archived, the completion tasks are:
  + insert a single row into the *METRICS\_DB.CrdsCatalog* table, and a row for each data file into the *METRICS\_DB.CrdsDataFiles* table (all inserts in a single transaction)
  + send a “successful delivery” email to the email addresses identified by the EMAIL\_LIST keyword
  + remove the temporary $DATA\_ROOT\_DIR/crds/temp/<catalog fileset>.pkl file
  + remove the data files from the delivery area
  + remove the original catalog file (\*.cat\_proc) from the delivery area

### Failures

If the timeout or any local processing error occurs (e.g., a database action fails), processing terminates with no cleanup.

HTCondor automatically creates a [rescue DAG](#rescue_dag), which knows how to pick up processing from where the error occurs.

# CRDS Metrics

Metrics about each CRDS delivery are stored in the JWST Data Processing Metrics database, in the CrdsCatalog and CrdsDataFile tables.

### CrdsCatalog Table

Each catalog/delivery gets a single row.

deliveryNumber: VARCHAR(8); primary key; CRDS delivery number, from the catalog file name; uniquely identifies a CRDS delivery

catalogName: VARCHAR(20); simple name (no path) of the manifest file that facilitated the delivery

receiptDate: DATETIME2; timestamp the catalog file was received in the local processing directory; UNIX file modification time to seconds (DATETIME2 requires microseconds; they will always be 0)

completionDate: DATETIME2; timestamp when the metrics data was inserted into the CrdsCatalog table, indicating completion of the catalog file processing; UNIX system time to seconds (DATETIME2 requires microseconds; they will always be 0)

### CrdsDataFile Table

Each data file (both reference and map types) gets a row in CrdsDataFile.

deliveryNumber: VARCHAR(8), 1st part of primary key; CRDS delivery number, from the catalog file name; links to CrdsCatalog; associates all of the data files in a CRDS delivery

fileName: VARCHAR(35); 2nd part of primary key; simple name (no path) of the data file

fileType: VARCHAR(3); “REF” for FITS files, “MAP” for CRDS mapping files

fileSize: BIGINT; data file size in bytes

archiveFileSet: VARCHAR(30); file set that identifies the file in the archive