CRDS Server Submissions,

Context Selection,

Reprocessing

# CRDS Server File Submissions

The ReDCaT team primarily makes two kinds of file submissions into CRDS:

1. New reference files
2. CRDS rmap updates to change reference file assignment

These deliveries are made using a combination of CRDS and ReDCaT scripts interacting with the CRDS server. The CRDS server can also be used independently of all command line tools and typically is for rmap updates.

ALL deliveries ultimately push files into the CRDS server where the files are certified before being accepted, where new rules are automatically generated, and where the resultant certify output and rules differences can be reviewed before confirming the delivery.

## CRDS Submission Tracking E-mails

During the submission process performed by ReDCaT, CRDS sends STARTED, READY, and CONFIRMED/CANCELED/FORCED e-mails to [crds-servers@stsci.edu](mailto:crds-servers@stsci.edu) and [redcat@stsci.edu](mailto:redcat@stsci.edu) as the submission progresses through the checking, generation, and review processes. These communicate to the broader team that a submission is under way and support monitoring and review.

A recent NIRISS delivery resulted in this concluding FORCED e-mail:

FORCED 'batch submit' by 'bbrooks'.

Final Results:

--------------

https://jwst-crds.stsci.edu//display\_result/0b999c02-414f-4a30-ada6-311b9c587677

Confirm/Cancel Url:

-------------------

https://jwst-crds.stsci.edu//display\_result/87ef134a-18fa-4bf6-9f22-308ab1493817

Description:

------------

The linearity reference file has new linearity coefficients and the new saturation reference file has new values in the science extension to match the linearity reference file. The distortion reference file has been updated with the correct 2D polynomial transformation from real to ideal pixels and back. The photom reference file values have been updated to include contamination along the light path of JWST for NIRISS. The drizzle paramters reference file has been newly created and is for calibration steps Level 2B and Level 3. Delivered by Michael A. Wolfe for NIRISS.

Uploaded Files:

----------------

niriss\_ref\_distortion.asdf --> jwst\_niriss\_distortion\_0008.asdf

niriss\_ref\_drizz\_par.fits --> jwst\_niriss\_drizpars\_0002.fits

niriss\_ref\_linearity.fits --> jwst\_niriss\_linearity\_0010.fits

niriss\_ref\_photom.fits --> jwst\_niriss\_photom\_0028.fits

niriss\_ref\_saturation.fits --> jwst\_niriss\_saturation\_0010.fits

Generated Files:

-----------------

jwst\_0417.pmap --> jwst\_0418.pmap

jwst\_niriss\_0092.imap --> jwst\_niriss\_0093.imap

jwst\_niriss\_distortion\_0014.rmap --> jwst\_niriss\_distortion\_0015.rmap

jwst\_niriss\_drizpars\_0002.rmap --> jwst\_niriss\_drizpars\_0004.rmap

jwst\_niriss\_linearity\_0011.rmap --> jwst\_niriss\_linearity\_0013.rmap

jwst\_niriss\_photom\_0015.rmap --> jwst\_niriss\_photom\_0017.rmap

jwst\_niriss\_saturation\_0008.rmap --> jwst\_niriss\_saturation\_0010.rmap

Reference file submitters typically “lock” the instrument they’re submitting files for on the CRDS server, preventing other simultaneous conflicting submissions for that instrument.

The designator FORCED indicates that CRDS instrument locking timed out or was voluntarily dropped by the time the submission was reviewed and OK’ed. The normal response is CONFIRMED, FORCED implies a need for extra ReDCaT team coordination to ensure no work is lost.

The “Uploaded Files” indicator demonstrates how CRDS renames incoming files named by developers into the standard CRDS form. Multiple references for multiple types of the same instrument can be delivered simultaneously.

The “Generated Files” section illustrates how delivered files result in the automatic update of the rmap associated with each delivered file type, as well as the generation of higher level context files for that instrument (.imap) and the overall pipeline (.pmap). The source rules files are typically automatically taken from a derivation or “edit” context based on the instrument and types being delivered. The “edit” context is typically the context of the previous delivery and often ahead of the operational context in use in the pipeline.

## CRDS Submission Monitoring and Review

The STARTED e-mail link directs the submitter to a web page that tracks submission progress in real time. Similar log messages are also delivered to the CRDS command line file submission client that is embedded in ReDCaT front-end file submission scripts.

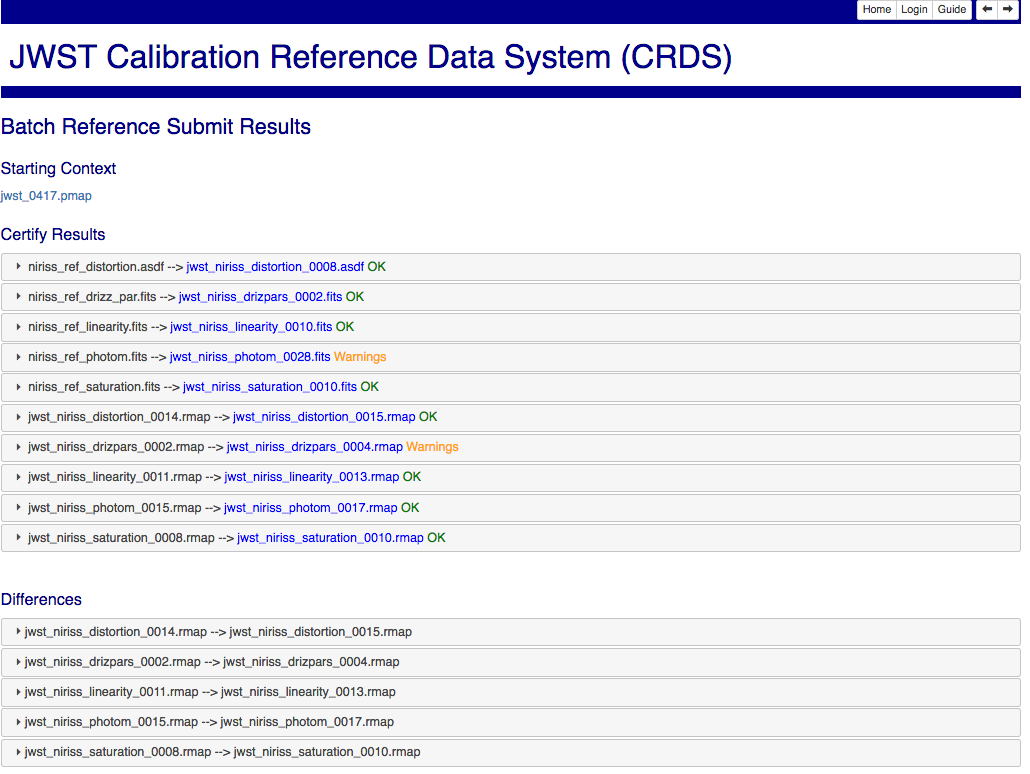
Submission processing can take several hours as potentially 50 gigabytes of references are certified, FITS checksums are verified, CRDS sha1sums are computed, new files are added to the CRDS rules, and preliminary CRDS database entries are created for each file.

## CRDS Submission Confirmation Results

When CRDS is finished preliminary file checking and rules generation, a READY e-mail is sent and a web page/link is available with a buttons to confirm or cancel the submission based on review.

The CRDS server directly performs complex checks based on (a) internally defined file keyword and format constraints (b) direct application of the JWST CAL data models and their constraints (c) fitsverify and astropy checks, as well as JSON and YAML syntax checks (d) differences to generated rules, which often reveal problems in newly developed reference file assignment criteria as teams learn how to create reference files and specify intended usage.

The outcome of submission processing is a page like this:



Each of the accordions shown opens up into more detailed information about that particular item.

The review and approval link is permanent and contains useful information on checks performed and how the rules were updated. Once a decision is made the buttons are removed and the decision is displayed instead.

Top of Form

# Delivery Protocol

Ultimately each of the files submitted to CRDS, or generated by CRDS, is stored on the CRDS server permanently and then linked to a delivery directory to initiate archiving. In addition to the links for delivered files, CRDS creates a catalog file that lists one delivered file per line.

Delivery between systems commences when the pipeline’s CRDS poller notices a new catalog file in the delivery directory and begins processing. At that point, the catalog is renamed to .cat\_proc. If the delivery fails, it will either be left as .cat\_proc or potentially renamed to .cat\_err as an indicator of status. If/when the delivery succeeds, every file link is removed, and finally, all forms of the .cat file are removed as well.

When the links and .cat file are finally removed, CRDS believes the delivery is complete and assumes the files are available from the archive. CRDS then marks the files as archived in CRDS, releasing them so that they can be sync’ed by pipelines and remote users.

## Shared Delivery Directory (CRDS server: CRDS Poller)

The CRDS archive delivery protocol revolves around the catalog files and shared delivery directory. Both CRDS and the archive CRDS poller mount the same storage.

For the JWST CRDS B-string the server mount point for the directory is:

/ifs/crds/jwst/bit/server\_files/deliveries

corresponding to shared disk volume:

isistor10e:/ifs/isifs/archive/public/jwst/bit

For JWST CRDS OPS the delivery directory is:

/ifs/crds/jwst/ops/server\_files/deliveries

for volume:

isistor10s:/ifs/isifs/archive/public/jwst/reference\_files\_ops

The *directory paths* can vary by (a) DMS string such as OPS, TEST, DEV, B-string I&T and (b) Virtual machine within the DMS such as the CRDS server or CRDS archiving poller.

The *volume* is conceptually a network disk drive and shared by both the CRDS server and the CRDS archiving poller machines to transfer files between systems. The network and storage partition can be reconfigured by ITSD without logically impacting those systems as long as the contents of the old volume are copied exactly. Changes to the network link (e.g. isistor10e) can dramatically impact delivery and sync performance.

## Delivery Failure Recovery

A frequent failure recovery process performed on the CRDS server is to reset files in the delivery directory preparing them for a second attempt by the pipeline’s CRDS poller and archive systems. This task can be performed by either side of the shared file system interface, but is generally performed within the CRDS server as follows:

$ ssh iljwdmsbcrds

% server # alias for “cd server source code directory”

% source env.csh

% deliveries # alias for “cd delivery directory”

% ls

…. check that only one delivery is pending, or… figure it out.

% rm –f \*

% catalogs # alias for “cd catalogs directory”

% redeliver\_catalogs –-catalogs jwst\_440.cat

… file links and catalog file are recreated in delivery directory…

Even after a delivery completes or fails, CRDS maintains internal copies of catalogs, generated files, and submitted files that can be relinked to the delivery directory to “reset” the delivery for a second attempt. The internal files are owned and writable only by the “crds” server user. The “crdsoper” group is able to remove the links from the delivery directory.

# Preview Testing by ReDCaT or CAL developers

An optional step possible once references are ingested and archived is to perform “preview” tests using the new files before switching the pipeline or extended JWST community to using them as the default.

Previewing is done by explicitly using the new context. For JWST generally by setting CRDS\_CONTEXT=jwst\_wxyz.pmap in the environment.

Files cannot be used by pipeline systems until synced as discussed below. It is not necessary (or sensible) to change the default context prior to syncing the pipeline if previews are desired. A subsequent context update and sync can be performed once testing has vetted a context. Generally previews are skipped for routine deliveries.

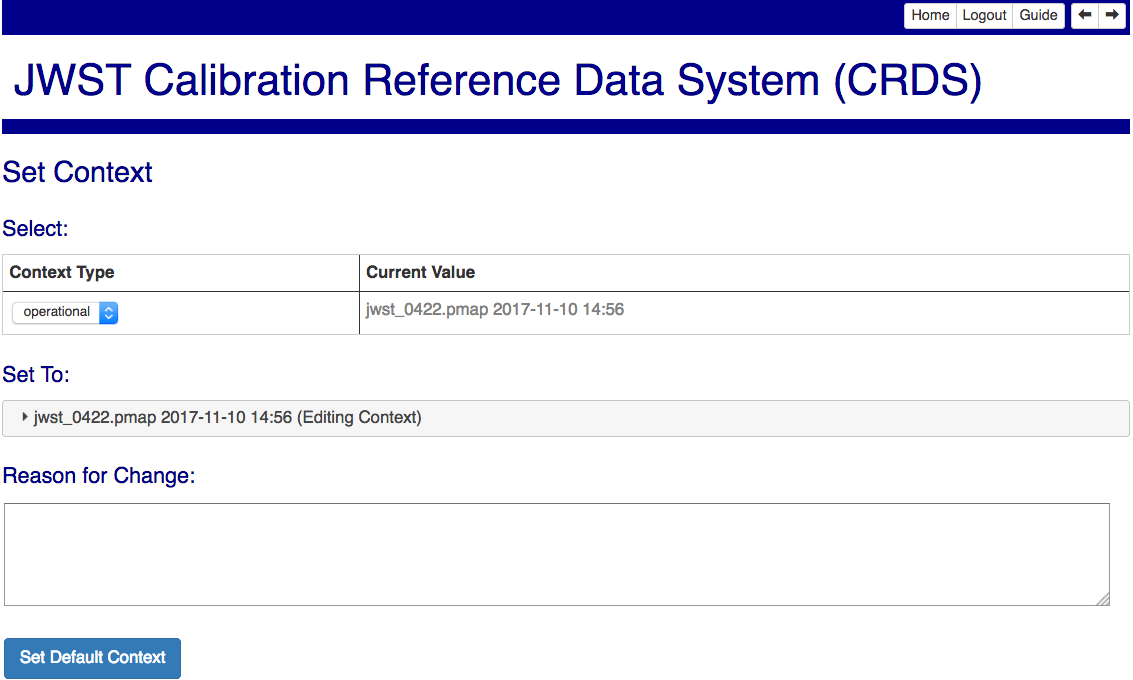
In the case of routine JWST development, files are currently delivered to the [JWST OPS CRDS Server](https://jwst-crds.stsci.edu/) and then automatically synced to the public read only cache at */grp/crds/cache*. At that stage files are broadly available, but typically the default is updated for all users only after the CAL development group has vetted the new rules and references against their current software by running nightly regression tests.

During JWST development, the CRDS OPS server stands in for the archive as the primary file store. Periodically during a build, the OPS server is mirrored to the I&T string that is under development. Unlike the OPS environment, I&T strings typically have full archive ingest and download support. I&T strings however do not perform the normal file submission processes, initialization of the archive is typically performed by resetting all deliveries using *redeliver\_catalogs.*

# Setting the Default Context

Once they approve of a delivery, the ReDCaT team e-mails an operator who logs into the CRDS server to use the Set Context function to designate the new context and newly delivered files as the default for everyone. This does not take effect in the pipeline until the pipeline cache is synced to the server. During JWST development, a similar approval and context update process is performed within SCSB. I&T strings are fully configured with archive support.

An operator updates the default context by going to the Set Context page:



For single context switches, jumping from context N to context N+1, CRDS normally populates the *Reason For Change* with the *Description* from the related file submission as a head start for the operator. For multi-context switches, e.g. from N to N+M, CRDS leaves the *Reason For Change* blank and the operator and ReDCaT team are required to devise a suitable summary.

The *Context Type* chosen by a pipeline operator should always be *operational*. The other choices are *edit* and *versions* which are used to control the default starting context for the next submission, and to define the context used to determine calibration software versions for the CRDS archive service.

Updating the default context is a consequential activity. It immediately results in a new entry in the CRDS Context History, and within a few minutes will trigger the CRDS reprocessing computation that determines the datasets impacted by the new reference files and/or rules.

# Synchronizing Pipeline CRDS Cache

After setting the new default context, the pipeline operator downloads the associated CRDS rules and references to the pipeline’s CRDS cache by running CRDS synchronization tools (*crds syn*c wrapped as *cron\_sync*) currently further wrapped by the pipeline’s *crds\_sync\_wrapper.csh*.

A cache sync can be performed even prior to Set Context as a way of pre-positioning large deliveries without using them. A cache sync **must** be performed after Set Context for the new default to take effect. Output from a successful run will generally include information about the size, source, and data rates for downloaded files. The trailing output from a recent rules-only small delivery is shown below:

$ crds\_sync\_wrapper.csh

….

2017-11-13 20:55:37,717 - CRDS - DEBUG - Connected to server and computing locally, updating CRDS cache config and operational context.

2017-11-13 20:55:37,731 - CRDS - DEBUG - CACHE updating: '/ifs/int/jwstb/ref/build7.1/crds/cache/config/jwst/server\_config'

2017-11-13 20:55:37,753 - CRDS - DEBUG - CACHE updating: '/ifs/int/jwstb/ref/build7.1/crds/cache/config/jwst/bad\_files.txt'

2017-11-13 20:55:37,777 - CRDS - DEBUG - CRDS JSON RPC push\_remote\_context (...) -->

2017-11-13 20:55:37,844 - CRDS - DEBUG - RPC OK

2017-11-13 20:55:37,844 - CRDS - INFO - Pushed cached operatonal context name 'jwst\_0422.pmap' to CRDS server

2017-11-13 20:55:37,844 - CRDS - INFO - STARTED 2017-11-13 20:54:45.04

2017-11-13 20:55:37,844 - CRDS - INFO - STOPPED 2017-11-13 20:55:37.84

2017-11-13 20:55:37,844 - CRDS - INFO - ELAPSED 0:00:52.80

2017-11-13 20:55:37,844 - CRDS - INFO - 6 total-files at 0.1 total-files-per-second

2017-11-13 20:55:37,844 - CRDS - INFO - 30.5 K total-bytes at 577.2 total-bytes-per-second

2017-11-13 20:55:37,845 - CRDS - INFO - 0 errors

2017-11-13 20:55:37,845 - CRDS - INFO - 0 warnings

2017-11-13 20:55:37,845 - CRDS - INFO - 12 infos

Key aspects of the output are “CACHE updating … server\_config” and “0 errors.” One other important issue is the context echo. Each operator must set up the required crypto key for the “Pushed cached operational context name….” echo to succeed. Issues with correctly setting up the crypto, or not setting up the crypto, have caused the crds\_sync\_wrapper.csh script to hang without output. This setup is intentionally left vague because it doubles as a lightweight form of authentication.

# Automatic Sync of /grp/crds/cache

CRDS is designed to support many identical caches that are similar to clones of a distributed version control system like git.

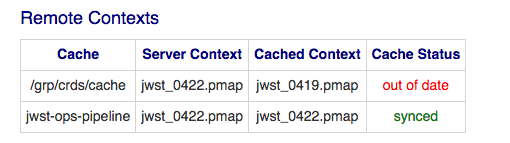
The pipeline has a private cache discussed above that is managed by operators and dedicated to pipeline use for the sake of hardware availability and capacity. It is normally read only during pipeline runs, and read-write when an operator syncs it to the server. This scheme makes the pipeline (a) independent of the CRDS server during ongoing calibrations (b) free of cache locking (at this time) and (c) safe for massive parallelism in the pipeline.

Other CRDS users and CAL typically use an identical read only copy of the CRDS cache located at /grp/crds/cache on the Central Store. /grp/crds/cache is slaved to the OPS server of both projects, and is not affected by I&T servers. It drives most onsite JWST development. Both HST and JWST update /grp/crds/cache automatically from the OPS servers using cron jobs currently running every 10 minutes. Similar to the pipeline cache, /grp/crds/cache is read only for CRDS users.

Remote users or users requiring read-write files are encouraged to set CRDS\_PATH and CRDS\_SERVER\_URL to define personal demand-based CRDS caches that transparently download references required for their specific datasets. This avoids the inefficiencies of repeated hidden downloads required by current VPN systems and makes it possible to experiment with rules and references.

# Verifying Cache Syncs and Context Updates

When the context push is successful, any authenticated user, typically a pipeline operator or file submitter, can log into the CRDS server and see sync status that has been echoed from a broadly used CRDS cache. It’s important to note that *anyone* can create their own complete or demand-based CRDS cache, but only critical shared caches are tracked here, not personal caches.



In the above picture *jwst-ops-pipeline* corresponds to the pipeline’s cache on any string; the name does not change between strings. Cached Context reflects what CRDS believes the pipeline is most likely using based on prior syncs and the echo. Here it indicates a successful sync for *jwst-ops-pipeline*.

The sync for */grp/crds/cach*e has not yet occurred for this context, while jwst\_0422.pmap is available and selected as the universal default, evidently /grp/crds/cache has not yet been sync’ed (or the sync failed) and is still using jwst\_0419.pmap by default. Indeed, syncing */grp/crds/cache* is only performed on the OPS servers, so I&T servers often lag permanently when Set Context is done on the I&T server rather than copied from OPS.).

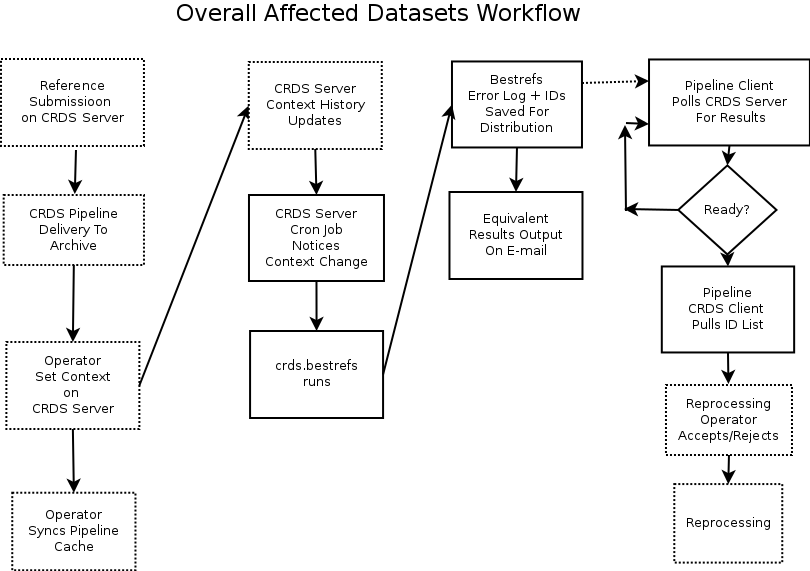
# Reprocessing

Based on the submission of new reference files, CRDS makes recommendations to the pipeline of which datasets would benefit from the new reference files and are candidates for reprocessing.

## Reprocessing Workflow

Starting from a ReDCaT file submission,

and concluding with the pipeline reprocessing affected datasets, the overall multi-system multi-team workflow is depicted here:



Solid boxes identify portions of the flow in which CRDS s/w directly related to reprocessing is involved in the workflow.

## Reprocessing Computation

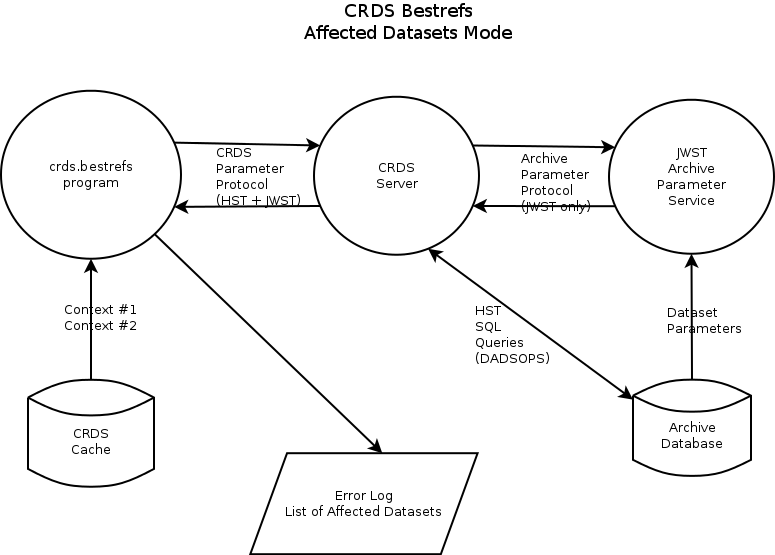
CRDS reprocessing recommendations are computed by a cron job running on the CRDS server that triggers the computation within 5 minutes of choosing a new default context.

The basis of the affected datasets computation is to run applicable dataset parameters through the CRDS bestrefs program on two different CRDS contexts and compare the results. Any dataset where some reference file changes is a candidate for reprocessing.

The reprocessing computation is implemented by one mode of the CRDS bestrefs program that fetches input parameters for large numbers of datasets from archive databases rather than reading smaller numbers of FITS headers. Unlike running bestrefs in the HST pipeline, the outcome of this mode is to print affected dataset IDs rather than update FITS headers with best references.

## Archive Parameter Source

As mentioned, reprocessing dataset parameters used to compute best references are retrieved in large numbers from archive databases.



crds.bestrefs is the CRDS command line tool that computes reprocessing recommendations. The crds.bestrefs tool uses a project-agnostic protocol to fetch parameters from the CRDS server. The CRDS server uses project specific protocols to access archive databases that define dataset best reference matching parameters.

For JWST it’s important to note that the archive provides a dedicated web service to fetch parameter sets. The protocol essentially has two parts CRDS can query:

1. Return the list of all IDs for an instrument after some date
2. Return the parameters for some list of IDs

For HST, CRDS has a direct interface to the DADSOPS (replicated) archive database and does it’s own SQL queries.

## Kinds of CRDS Errors

To understand CRDS reprocessing errors, a broader view of possible CRDS errors is useful:

1. CRDS can fail to find a reference file
2. CRDS can find the wrong reference file
3. CRDS can match multiple reference files due to bad rules
4. A reference file can have errors that cause calibrations to fail

For CRDS reprocessing, (1) and (2) are most relevant and can be collectively classified as “lookup errors”. Lookup errors in turn can be caused by:

1. A problem with the archive’s parameters for some dataset
2. A problem with the CRDS rules
3. Missing reference file coverage

Discerning between parameters set errors and rules errors is important for debugging reprocessing.

## Error Logging

Each dataset is generally assigned 15-20 reference files using criteria that vary by instrument and reference type and often involve many different cases to cover different instrument modes and specific configurations. During JWST development, working with thousands of new and changing datasets interacting with new CRDS rules and references, errors are expected and common.

As each dataset parameter set is processed by crds.bestrefs, specific log errors are issued type-by-type, file-by-file, as they occur, for both old and new contexts. To simplify post-processing error analysis, CRDS summarizes error messages into classes of similar problems and presents those in the log output as *unique error classes.*

The nature of CRDS rules corrections is that new versions of rules are created and older versions of rules remain exactly as-is. This means that unless dataset parameters change, an old context will continue to produce exactly the same errors indefinitely. In particular, an old context will definitely produce errors during the reprocessing run in which the new context provides fixes for those same errors.

## Narrowing Reprocessing Computations

Since HST has over a million dataset IDs (as seen by CRDS), brute force computations for two contexts for all instruments originally took on the order of 12 hours to complete. Consequently, as an optimization, the difference between contexts has been exploited to determine (a) possibly affected instruments (b) possibly affected reference types and (c) possibly affected observation dates based on affected USEAFTER dates. Collectively, those three pieces of information are typically used to reduce the datasets that are examined to a few thousand enabling the computation of datasets to reprocess to complete in minutes.

The reprocessing log includes information on the datasets and aspects of the contents that are examined, and team members can eyeball those for consistency with the delivery.

## Saving Repro Results

When the CRDS server finishes the crds.besrefs repro computation, it saves both the program log and the list of affected IDs permanently. CRDS also issues an e-mail at that time to containing snipped log output and a compressed attachment listing all the affected IDs.

## Retrieving Repro Results

CRDS provides a repro client script that has been customized for integration with the pipeline environment by SDP. The pipeline operator uses that integration script (crds\_impact.csh) to fetch the stored results from the CRDS server. Prior to the completion of the CRDS Server computation, an attempt to retrieve the results should produce a “no results available” log message of some kind. When results are complete, the client script can be used to retrieve both affected IDs and the complete reprocessing log so that any errors can be reviewed.

# Reviewing Team Activity

## Audit Trail

CRDS keeps an audit trail of confirmed deliveries and context changes on the server, as well as changes in state like marking files bad.

Following this link can display activity:

[General Activity for Pmaps](https://jwst-crds.stsci.edu/recent_activity_query/)

Specific kinds of activity can be reviewed by supplying additional criteria here:

[More Specialized Activity Queries](https://jwst-crds.stsci.edu/recent_activity_input/)

Pursuing file links can provide additional information on those files. Most important events result in the creation or selection of a pmap so the default query lists “any activity related to a pmap”. More specialized queries can also be constructed, e.g. any activity related to a specific instrument, type, or file.

## Seeing Context History and Differences

The top level CRDS page displays the last 4 operational contexts in use in the pipeline and a brief description of each and the date of activation.

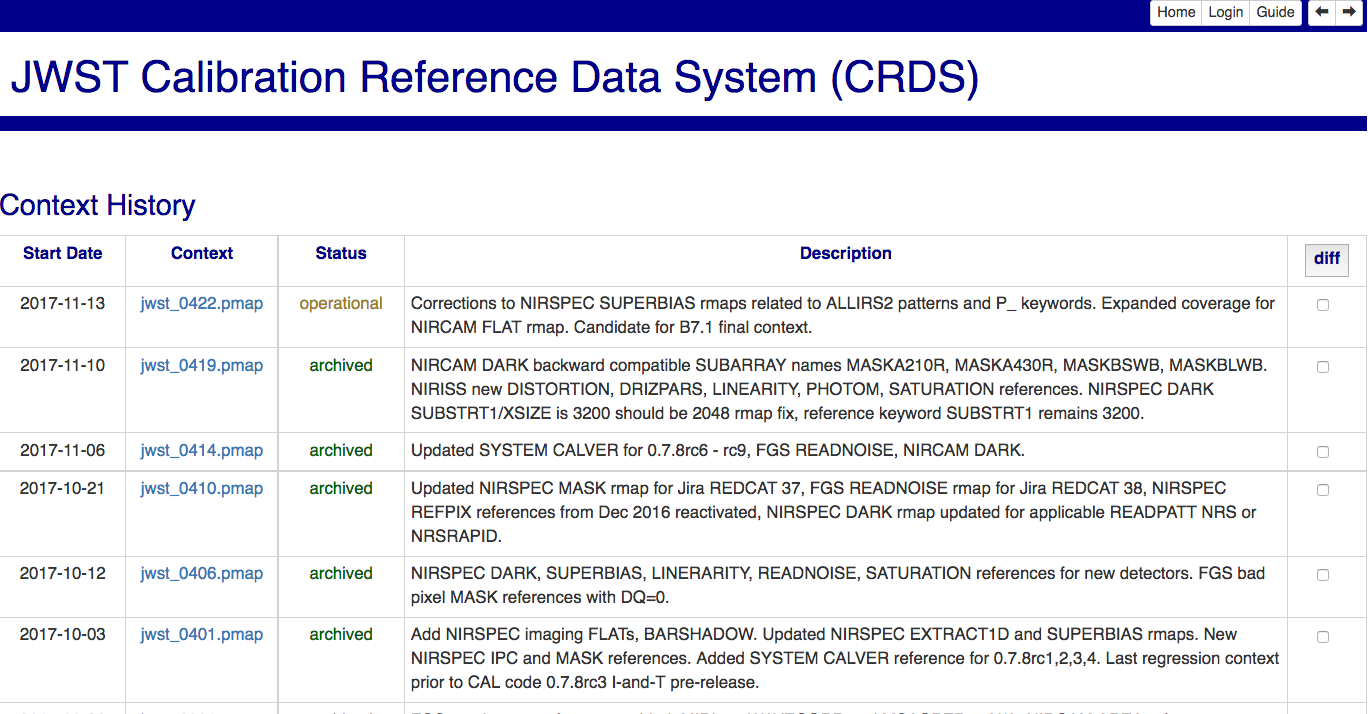
This link:

[Context History](https://jwst-crds.stsci.edu/display_context_history/)

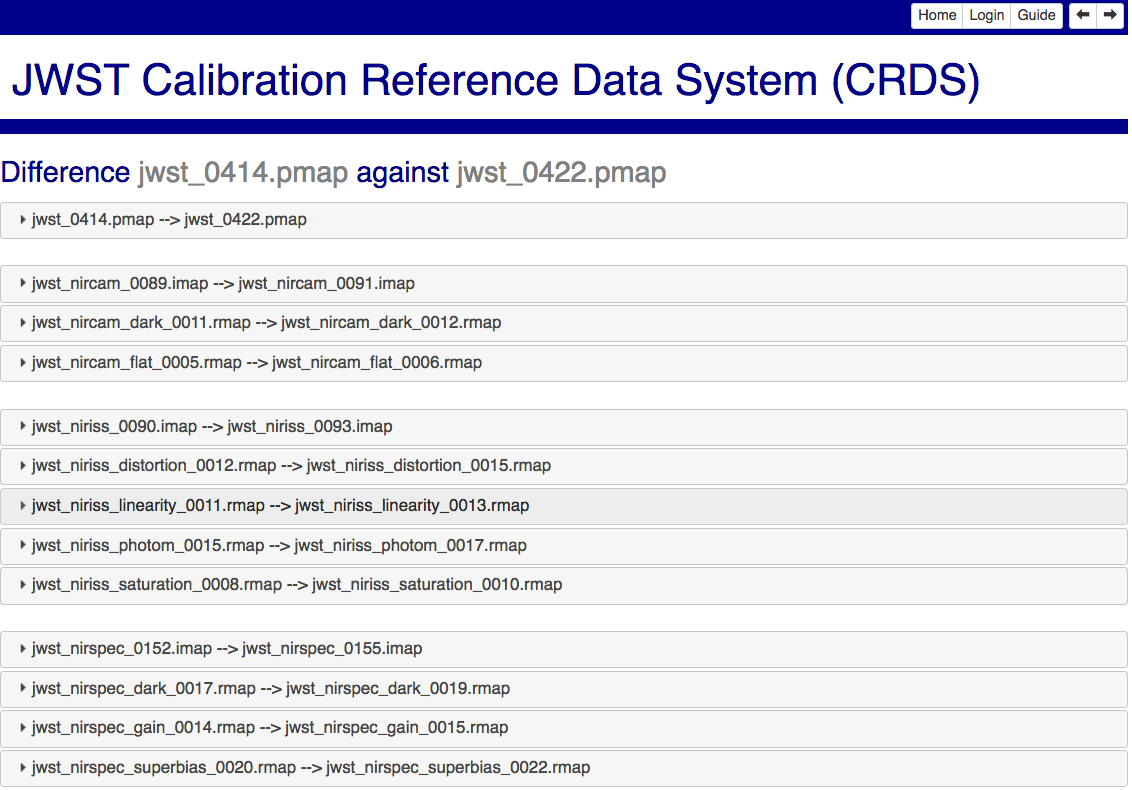
displays the history of all contexts ever selected as the pipeline default and when they became active or were superseded.

Particularly for JWST, the operational context frequently skips through several .pmap files so the context does not always change from N to N+1.

The *Context History* looks similar to this and extends backward in time through all previously default contexts:



After selecting the diff checkbox of two contexts, the diff button can be clicked to generate a page shown here for 414 -> 422:



Each accordion panel can be opened to show both the logical and textual differences between those two versions of rules files:

