Build 5 DMS Level Calibration Reference Data System Requirements Verification

David Grumm and Todd Miller
8/27/2015

Table of Contents

Introduction
1 DMS-538: The CRDS shall commit new reference files without requiring their use in operations
2 DMS-541: The CRDS shall have the capability to recreate the list of appropriate calibration reference files for a point in the past given the date of the query, version of software, and a specific data set
3 DMS-542: The CRDS shall record meta data about the reference files and all information regarding how data sets are matched to reference files including (but not limited to): How the reference file was generated (or links to documents regarding that); Who created the reference file; Who committed the reference file; Information about the significance of the change
4 DMS-543: It shall be possible to mark committed reference files as bad so they will no longer be selected by CRDS
5 DMS-545: A tool shall be provided that will list where differences exist for recommended reference files between two rules and list of data sets
6 DMS-546: Reference files shall include in their headers information in the form of header comments that describe the procedure for making the reference file or point to a reference that describes that procedure
7 DMS-547: CRDS shall provide a tool to show the set of active reference files being used for all supported versions of the software, or a specific version of the software
8 DMS-548: CRDS shall provide a tool to show active files associated with specific instrument observing modes
9 DMS-641: CRDS shall have the capability to mark a specific rule as bad so as to prevent its use by the CRDS
10 DMS-642: CRDS shall have the capability to undo the use of a specific rule mapping data to a reference file
11 DMS-669: The CRDS catalog shall be capable of being queried read-only through a web-based programmatic service
12 DMS-680: CRDS shall validate the format of required input files
13 DMS-681: All CRDS reference files shall have unique names
14 DMS-682: All CRDS rules and context files shall have unique names

Introduction:

The purpose of this document is to detail the verification of each of the Build 5 DMS Level CRDS requirements. The Integration and Testing Branch should follow these steps for verification. For each of the 13 requirements, information from the spreadsheet XXX is given: the Requirement ID, the DMS Level 4 Requirements, the Verification Method (Demonstration, Test, or Inspection), Comments, and the steps required.

For verifying these requirements, the Integration and Testing Branch will need to reset these CRDS environmental variables for the new CRDS servers:

```
% setenv CRDS_SERVER_URL  https://jwst-crds-b5it.stsci.edu
% setenv CRDS_PATH  $HOME/crds_cache_b5it
% python -m crds.sync --all   # clones all CRDS rules to $CRDS_PATH and subdirs
```

The needed input files and parameter files have been copied to the directory TBD. The tester will need to copy these files to a working directory to verify many of the steps.

1 Requirement ID: DMS-538

DMS Level 4 Requirement:

The CRDS shall commit new reference files without requiring their use in operations.

Verification Method:
Test

Comments:

Verification Procedure:

As part of submitting reference files to CRDS a new version of reference file assignment rules is generated which will assign the new files. However, the default context being used does not immediately change to the new version of rules. Hence, files “in” CRDS do not immediately go into use by default. For JWST, files not assigned by the default rules can be tested in advance by specifying the version of CRDS rules via the CRDS_CONTEXT environment variable. After advance checkout, the new rules are made the default by using the Set Context page on the CRDS server. CRDS is a distributed system with a central repository much like a version control system. Once the default context is set on the central server, it is propagated to remote pipelines using the CRDS cron_sync script run in the pipeline system.

1. Run the flat field calibration step using the dataset jw00001001001_01101_00001_MIRIMAGE_assign_wcs.fits, having CRDS automatically fetch the reference file using the default context:

% strun flat_field.cfg jw00001001001_01101_00001_MIRIMAGE_assign_wcs.fits

2. Create a directory to hold the output:
mkdir default_context

3. Copy the resulting output file and log file there:
cp jw00001001001_01101_00001_MIRIMAGE_assign_wcs_flat_field.fits pipeline.log default_context

4. Reset the context to a previous value (simulating a file delivery, in reverse).

% setenv CRDS_CONTEXT jwst_0012.pmap

5. Redo step 1, using this earlier context:
strun flat_field.cfg jw00001001001_01101_00001_MIRIMAGE_assign_wcs.fits

6. Create a directory to hold the output:
mkdir old_context

7. Copy the resulting output file and log file there:
% cp jw00001001001_01101_00001_MIRIMAGE_assign_wcs_flat_field.fits pipeline.log old_context

8. The two sets of results can be compared by using fitsdiff on the data products, and by confirming that different flat reference files were used by examining the log files.

2 Requirement ID: DMS-541

DMS Level 4 Requirement:

The CRDS shall have the capability to recreate the list of appropriate calibration reference files for a point in the past given the date of the query, version of software, and a specific data set.

Verification Method:
Test

Comments:

Verification Procedure:

To satisfy the requirement, the tester can invoke the crds.bestrefs program using a date based context specification and a FITS dataset.

For example:

% python –m crds.bestrefs --new-context jwst-2015-08-05 --print-new-references --files my_dataset.fits

The date based spec jwst-2015-08-05 is translated via the CRDS context history on the server into the latest context preceding or equal to that date, which in this case is: jwst_0063.pmap. The reference files recommended by that context for my_dataset.fits are printed out as a consequence of the --print-new-references switch.

3 Requirement ID: DMS-542

DMS Level 4 Requirement:

The CRDS shall record meta data about the reference files and all information regarding how data sets are matched to reference files including (but not limited to): How the reference file was generated (or links to documents regarding that); Who created the reference file; Who committed the reference file; Information about the significance of the change.

Verification Method:
Demonstration

Comments:

Verification Procedure:

When the tester submits a reference file, he specifies the method that he used to generate the file using the file's HISTORY and DESCRIP parameters. On the website, the submitter specifies the AUTHOR or author's in the creator field on the web form; the submitter's username is recorded automatically. The web form Change Level field records the significance of the change. The web form Description field can be used to include additional information as free form text.

Requirement ID: DMS-543

DMS Level 4 Requirement:

It shall be possible to mark committed reference files as bad so they will no longer be selected by CRDS.

Verification Method:
Demonstration

Comments:

CRDS maintains the invariant that the operational context does not contain bad files. Consequently, the first step in marking reference or rules files as bad is to switch to a context that does not contain the soon-to-be bad files. This may entail creating the new context by some method prior to switching, perhaps by deleting or replacing files in the current version of operational rules. An alternative to creating a new clean context is switching to an older clean context which did not contain the prospective references or any other bad files.

Merely containing bad references does not taint all of the rules which point to the bad references; those contexts can still be used for other cases without generating errors or warnings as long as the bad references are not actually assigned. Even with this lesser restriction, CRDS does not allow setting these contexts as default.

Verification Procedure:

1. Switch to an older default context not containing the bad files using the website Set Context feature. This is an alternative to creating a new context which does not contain the bad files and switching to that instead.
2. Mark the files bad on the web site using Mark Files Bad.
3. Set CRDS_CONTEXT to the context which contains the bad files.
4. Run the pipeline on a dataset which assigns the bad files.
5. Verify that CRDS generates a hard error and the pipeline process exits with an error status of 1 indicating process failure.
6. Unset CRDS_CONTEXT so the default context is used.
7. Run the pipeline on the same dataset, the new context should assign alternative files.
8. Verify that alternate good files were assigned by the current default context.

4 Requirement ID: DMS-545

DMS Level 4 Requirement:

A tool shall be provided that will list where differences exist for recommended reference files between two rules and list of data sets.

Verification Method:
Test

Comments:

Verification Procedure:
The tester can invoke the crds.bestrefs program to list differences between recommended reference files by providing the names of the datasets using the optional ‘files’ argument.

Here is an example for two datasets:

% python -m crds.bestrefs -o jwst_0061.pmap -n jwst_0073.pmap --files jw00035001001_01101_00001_MIRIMAGE_uncal.fits jw00034001001_01101_00001_NIRISS_uncal.fits

With the following (edited) output:

CRDS : INFO No file header updates requested; dry run.
CRDS : WARNING UserWarning : jwst_lib.models.schema : meta.visit.total_exposures: '#TODO' is not of type u'integer'. Setting to default of None
CRDS : WARNING UserWarning : jwst_lib.models.schema : meta.visit.internal_target: '#TODO' is not of type u'boolean'. Setting to default of None
CRDS : WARNING UserWarning : jwst_lib.models.schema : meta.visit.external_target: '#TODO' is not of type u'boolean'. Setting to default of None
CRDS : WARNING UserWarning : jwst_lib.models.schema : meta.visit.too_visit: '#TODO' is not of type u'boolean'. Setting to default of None

< snip >

CRDS : WARNING UserWarning : jwst_lib.models.schema : meta.time.heliocentric_expstart: '' is not of type u'number'. Setting to default of None
CRDS : WARNING UserWarning : jwst_lib.models.schema : meta.time.heliocentric_expend: '' is not of type u'number'. Setting to default of None
CRDS : WARNING UserWarning : jwst_lib.models.schema : meta.time.heliocentric_expmid: '' is not of type u'number'. Setting to default of None
CRDS : INFO ===> Processing jw00034001001_01101_00001_NIRISS_uncal.fits
CRDS : ERROR instrument='NIRISS' type='AREA' data='jw00034001001_01101_00001_NIRISS_uncal.fits' :: New: Bestref FAILED: parameter='META.INSTRUMENT.PUPIL' value='GR700XD' is not in ['CLEARP']
CRDS : INFO instrument='NIRISS' type='DARK' data='jw00034001001_01101_00001_NIRISS_uncal.fits' :: New best reference: 'jwst_niriss_dark_0004.fits' --> 'jwst_niriss_dark_0005.fits' :: Would update.
CRDS : INFO instrument='NIRISS' type='FLAT' data='jw00034001001_01101_00001_NIRISS_uncal.fits' :: New best reference: 'jwst_niriss_flat_0002.fits' --> 'jwst_niriss_flat_0003.fits' :: Would update.
CRDS : INFO instrument='NIRISS' type='GAIN' data='jw00034001001_01101_00001_NIRISS_uncal.fits' :: New best reference: 'jwst_niriss_gain_0000.fits' --> 'jwst_niriss_gain_0001.fits' :: Would update.
CRDS : INFO instrument='NIRISS' type='IPC' data='jw00034001001_01101_00001_NIRISS_uncal.fits' :: New best reference: 'jwst_niriss_ipc_0001.fits' --> 'jwst_niriss_ipc_0002.fits' :: Would update.
CRDS : INFO instrument='NIRISS' type='LINEARITY' data='jw00034001001_01101_00001_NIRISS_uncal.fits' :: New best reference: 'jwst_niriss_linearity_0004.fits' --> 'jwst_niriss_linearity_0005.fits' :: Would update.

CRDS : INFO   instrument='NIRISS' type='MASK' data='jw00034001001_01101_00001_NIRISS_uncal.fits'
:: New best reference: 'jwst_niriss_mask_0002.fits' --> 'jwst_niriss_mask_0004.fits' :: Would update.
CRDS : INFO   instrument='NIRISS' type='PHOTOM'
data='jw00034001001_01101_00001_NIRISS_uncal.fits' :: New best reference:
'jwst_niriss_photom_0016.fits' --> 'jwst_niriss_photom_0017.fits' :: Would update.
CRDS : INFO   instrument='NIRISS' type='READNOISE'
data='jw00034001001_01101_00001_NIRISS_uncal.fits' :: New best reference:
'jwst_niriss_readnoise_0000.fits' --> 'jwst_niriss_readnoise_0001.fits' :: Would update.
CRDS : INFO   instrument='NIRISS' type='SATURATION'
data='jw00034001001_01101_00001_NIRISS_uncal.fits' :: New best reference:
'jwst_niriss_saturation_0004.fits' --> 'jwst_niriss_saturation_0005.fits' :: Would update.
CRDS : INFO   instrument='NIRISS' type='SUPERBIAS'
data='jw00034001001_01101_00001_NIRISS_uncal.fits' :: New best reference:
'jwst_niriss_superbias_0001.fits' --> 'jwst_niriss_superbias_0003.fits' :: Would update.
CRDS : ERROR   instrument='NIRISS' type='THROUGHPUT'
data='jw00034001001_01101_00001_NIRISS_uncal.fits' :: Old: Bestref FAILED:
parameter='META.INSTRUMENT.FILTER' value='CLEAR' is not in ['F277W', 'F380M', 'F430M', 'F480M']
CRDS : ERROR   instrument='NIRISS' type='THROUGHPUT'
data='jw00034001001_01101_00001_NIRISS_uncal.fits' :: New: Bestref FAILED:
parameter='META.INSTRUMENT.FILTER' value='CLEAR' is not in ['F277W', 'F380M', 'F430M', 'F480M']
CRDS : INFO   ===> Processing jw00035001001_01101_00001_MIRIMAGE_uncal.fits
CRDS : ERROR   instrument='MIRI' type='FRINGE'
data='jw00035001001_01101_00001_MIRIMAGE_uncal.fits' :: Old: Bestref FAILED:
parameter='META.INSTRUMENT.DETECTOR' value='MIRIMAGE' is not in ['MIRIFULONG',
'MIRIFUSHORT']
CRDS : ERROR   instrument='MIRI' type='FRINGE'
data='jw00035001001_01101_00001_MIRIMAGE_uncal.fits' :: New: Bestref FAILED:
parameter='META.INSTRUMENT.DETECTOR' value='MIRIMAGE' is not in ['MIRIFULONG',
'MIRIFUSHORT']
CRDS : ERROR   instrument='MIRI' type='STRAYMASK'
data='jw00035001001_01101_00001_MIRIMAGE_uncal.fits' :: Old: Bestref FAILED:
parameter='META.INSTRUMENT.DETECTOR' value='MIRIMAGE' is not in ['MIRIFUSHORT']
CRDS : ERROR   instrument='MIRI' type='STRAYMASK'
data='jw00035001001_01101_00001_MIRIMAGE_uncal.fits' :: New: Bestref FAILED:
parameter='META.INSTRUMENT.DETECTOR' value='MIRIMAGE' is not in ['MIRIFUSHORT']

CRDS : INFO   7 errors
CRDS : INFO   72 warnings
CRDS : INFO   13 infos

5 Requirement ID: DMS-546

DMS Level 4 Requirement:

Reference files shall include in their headers information in the form of header comments that describe the procedure for making the reference file or point to a reference that describes that procedure.

Verification Method: Inspection

Comments:

Verification Procedure:

The tester can invoke crds.certify using the reference file as the argument, and setting the 'dump-provenance' optional argument, which will display the provenance keywords (DESCRIP, PEDIGREE, AUTHOR, USEAFTER, and HISTORY) from the header of the reference file.

For example:
python -m crds.certify --dump-provenance jwst_niriss_flat_0003.fits

Will result in:

```
CRDS  : INFO    ########################################


CRDS  : INFO    Certifying '/grp/crds/cache/references/jwst/jwst_niriss_flat_0003.fits' (1/1) as 'FITS' relative to context 'jwst_0074.pmap'


CRDS  : INFO    FITS file 'jwst_niriss_flat_0003.fits' conforms to FITS standards.
CRDS  : INFO    [0] DESCRIP This is a pixel flat reference file.------------------------------
CRDS  : INFO    [0] HISTORY This file is being delivered because it is the pixel flat reference file
CRDS  : INFO    [0] HISTORY . This file was created from the python scripts convert_niriss_pxl_flt_c
CRDS  : INFO    [0] HISTORY mdv.py and ref_pxl_flt.py and fits file provided by Kevin Volk. Document
CRDS  : INFO    [0] HISTORY ation describing the strategy and algorithms is found in "Description of
CRDS  : INFO    [0] HISTORY  the NIRISS Pixel Flat Reference File." Python scripts can be found at /
CRDS  : INFO    [0] HISTORY witserv/data17/mwolfe/niriss/test_comdev/test/test_pixel_flat and there
CRDS  : INFO    [0] HISTORY is no version number for the scripts. The data used to create the refere
CRDS  : INFO    [0] HISTORY nce file are:
CRDS  : INFO    [0] HISTORY frame35signal1230nm_pixelflat1.fits
CRDS  : INFO    [0] HISTORY frame35signal1230nm_pixelflat1_dq1.fits
CRDS  : INFO    [0] HISTORY frame35signal1230nm_pixelflat1_error.fits
CRDS  : INFO    [0] PEDIGREE GROUND
CRDS  : INFO    [0] USEAFTER Jan 01 2015 00:00:00
CRDS  : INFO    META.INSTRUMENT.DETECTOR = 'NIS'
CRDS  : INFO    META.INSTRUMENT.FILTER = 'CLEAR'
CRDS  : INFO    ########################################

CRDS  : INFO    0 errors
CRDS  : INFO    0 warnings
CRDS  : INFO    20 infos
```

6 Requirement ID: DMS-641

DMS Level 4 Requirement:

CRDS shall have the capability to mark a specific rule as bad so as to prevent its use by the CRDS.

Verification Method:
Test

Comments:

Verification Procedure:

CRDS maintains the invariant that the current operational context can never contain bad rules. Consequently, the process of marking any rule or reference bad requires first generating or reverting to a context which does not include the candidate bad files. This can be done by submitting replacements, or by submitting rules which delete the bad files. Once a clean context has been created and set as the default, it is then possible to use the Mark Files Bad page on the server to taint the bad files.

Different methods can be used to achieve a clean default context, that is, a context which WILL be clean after a rule is marked bad. For example, to eliminate a defective rmap, do the following:

1. Obtain a copy of the current (bad) rmap either from the cache or by browsing to the rmap's file details page on the web site and clicking the download link.

2. Modify the rmap as necessary to produce a good one using emacs.

3. Submit the modified rmap to the website using the Submit Mappings page with Generate Contexts selected. Generate Contexts can only be used for submitting new rmaps. This adds the new rmap, and derives a new imap and pmap which reference it.

4. Use Set Context on the web site to make the new clean context the operational default.

5. Use Mark Files Bad on the web site to mark the defective rules files as bad. Enter a description explaining why the file is designated as scientifically invalid. Any higher level rules files which reference the bad rules are tainted by inheritance and also become bad.

6. Synchronize remote caches using the cron_sync script or crds.sync tool or just by running single threaded calibration code to distribute the knowledge of the new context and bad files.

Requirement ID: DMS-642

DMS Level 4 Requirement:
CRDS shall have the capability to undo the use of a specific rule mapping data to a reference file.

Verification Method:
Test

Comments:

Verification Procedure:

The tester can test this requirement by setting the CRDS_CONTEXT environmental variable to define his private context to one predating that reference file.

1. Run pipeline using current context and capture resulting dataset with recorded references.
2. setenv CRDS_CONTEXT to an older context, one which precedes the addition of the undone file. Save the output dataset.
3. Run the pipeline again using the older context, save the output dataset.
4. Run fitsdiff on the datasets verifying that the reference file was undone by switching to the older context.

This requirement is also satisfied by the capability to generate a new version of CRDS rules omitting or replacing the reference file being “undone”.

1. Download the rmap for the reference file which is to be “undone”.
2. Change the rmap to omit or replace the undone reference.
3. Run crds.checksum and crds.certify on the rmap to seal the rmap and screen out basic errors.
4. Submit the modified rmap to the web site using the Submit Mappings page, generating a new context.
5. Run the pipeline with the first context.
6. Run the pipeline with the second context using CRDS_CONTEXT or the Set Context web page.
7. fitsdiff the output datasets from the two pipeline runs and verify that the undone file changed.

7 Requirement ID: DMS-669

DMS Level 4 Requirement:

The CRDS catalog shall be capable of being queried read-only through a web-based programmatic service.

Verification Method:
Demonstration

Comments:
From the spreadsheet (by unknown author):
“This means that we can run queries on the catalog (e.g., list all MIRI rules files for darks). Or list all MIRI dark files that have ever been committed and are not marked as unusable. At least be able to make queries that don't affect the state of the catalog (no modifications permitted). “

Verification Procedure:

1. List all MRI rules files for darks

% python -m crds.sql "select name from crds_jwst_catalog where instrument='miri' and filekind='dark' and type='mapping'" --update”
('jwst_miri_dark_0000.rmap',)
('jwst_miri_dark_0001.rmap',)
('jwst_miri_dark_0002.rmap',)
...
('jwst_miri_dark_0009.rmap',)

2. List all MIRI dark files that have ever been committed and are not marked as unusable

% python -m crds.sql 'select name from crds_jwst_catalog where instrument=”miri” and filekind="dark" and not blacklisted and not rejected and (state=”operational” or state=”archived”)'

Additional features of crds.sql can be seen using the --help switch, e.g. listing available tables and columns.

3. The crds.sql tool is a convenience script which wraps access to the CRDS catalog database. The tester can also download and locate a version of the CRDS catalog for use with sqlite3 as follows:

% python -m crds.sql --update --list-database-path
CRDS : INFO Downloading CRDS catalog database file.
CRDS : INFO Sqlite3 database file downloaded to:
/Users/jmiller/crds_cache_b5it/config/jwst/crds_db.sqlite3

% sqlite3 /Users/jmiller/crds_cache_b5it/config/jwst/crds_db.sqlite3
SQLite version 3.7.10 2012-01-16 13:28:40
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite>

The sqlite3 program is distributed with the sqlite3 software and Ureka and is independent of CRDS. Running sqlite3 independently provides access to all available sqlite3 features, not all of which are accessible under crds.sql.

CRDS reference file assignment criteria are stored in CRDS rules and hence are not accessible using the CRDS metadata catalog.

Requirement ID: DMS-680

DMS Level 4 Requirement:

CRDS shall validate the format of required input files.

Verification Method:
Demonstration

Comments:

Verification Procedure:

This step can be validated by invoking the command-line script crsd.certify to check a reference or mapping file against constraints on legal matching parameter values. This command line tool also performs checks of the FITS format and, when given a context, will compare the given file against the file it replaces looking for new or missing table rows.

For example, for a candidate reference file ‘jwst_niriss_photom_0017.fits’ in the current working directory:

```
> python -m crds.certify ./jwst_niriss_photom_0017.fits
CRDS  : INFO     [2015-09-04 14:56:08,898]  ########################################
CRDS  : INFO     [2015-09-04 14:56:08,899]  Certifying './jwst_niriss_photom_0017.fits' (1/1) as 'FITS' relative to context 'jwst_0074.pmap'
CRDS  : INFO     [2015-09-04 14:56:08,977]  FITS file 'jwst_niriss_photom_0017.fits' conforms to FITS standards.
CRDS  : INFO     [2015-09-04 14:56:09,647]  ########################################
CRDS  : INFO     [2015-09-04 14:56:09,647]  0 errors
CRDS  : INFO     [2015-09-04 14:56:09,648]  0 warnings
CRDS  : INFO     [2015-09-04 14:56:09,648]  4 infos
```

The format of the FITS file has been validated, relative to the current default context 'jwst_0074.pmap'.

As another example, for the mapping jwst_0043.pmap:
python -m crds.certify jwst_0043.pmap

omm: > python -m crds.certify jwst_0043.pmap

```
CRDS  : INFO     No comparison context specified or specified as 'none'.  No default context for all mappings or mixed types.
CRDS  : INFO     ########################################
CRDS  : INFO     Certifying '/grp/crds/cache/mappings/jwst/jwst_0043.pmap' (1/62) as 'MAPPING' relative to context None
CRDS  : INFO     ########################################
CRDS  : INFO     Certifying '/grp/crds/cache/mappings/jwst/jwst_fgs_0008.imap' (2/62) as 'MAPPING' relative to context None
CRDS  : INFO     ########################################
```

< snip >

```
CRDS  : INFO     Certifying '/grp/crds/cache/mappings/jwst/jwst_nirspec_saturation_0001.rmap' (61/62) as 'MAPPING' relative to context None
CRDS  : INFO     ########################################
CRDS  : INFO     Certifying '/grp/crds/cache/mappings/jwst/jwst_nirspec_specwcs_0003.rmap' (62/62) as 'MAPPING' relative to context None
CRDS  : INFO     ########################################
CRDS  : INFO     0 errors
CRDS  : INFO     0 warnings
CRDS  : INFO     126 infos
```

In this case, crds.certify recursively certifies all 62 sub-mappings which are taken from the CRDS cache verifying that the entire network of files successfully loads and validates.

8 Requirement ID: DMS-681

DMS Level 4 Requirement:

All CRDS reference files shall have unique names.

Verification Method:
Demonstration

Comments:

Verification Procedure:

To verify this requirement, the tester must have an account on the CRDS website. For each case below, the tester should first login to CRDS and navigate to the reference file batch submission page. When logging in, the submitter should choose the “niriss” lock. Test files are located at /grp/crds/aux/jwst/build5_test_files/dms_681. Three variables affect the execution of this requirement: submitted file name, submitted file contents (as digested using sha1sum), auto-rename option checked or unchecked. Files with identical names are rejected to satisfy this requirement. Files with identical contents are assumed to be erroneous re-submissions and rejected as an implementation detail; nevertheless it affects testing so it bears explanation. Auto-rename is a feature which controls automatic renaming of files, a web page check box enabling automatic renaming to be bypassed. For the sake of this testing, all examples will utilize files with unique contents, the focus will be on naming. Non-unique files are rejected prior to naming. Not all file names are acceptable if renaming is disabled.

Case 1) – submitted file has a new name and auto-rename is selected
1. Upload the file to submit 'my_ref1.fits'.
2. Enter the file creator/author name and a text description.
3. Submit the file and confirm

Expected outcome: this should result in the file having a new unique name auto-generated, and the file accepted. Since the file is being renamed, most reasonable upload names are acceptable.
Cleanup: none

Case 2) - submitted file has identical name to existing file, different contents, and auto-rename is deselected
1. Upload the file to submit 'jwst_niriss_mask_0004.fits'.
2. Enter the file creator/author name and a text description.
3. Submit the file.

Expected outcome: this file is rejected because it has a non-unique name even though it does have unique contents. The test version of jwst_niriss_mask_0004.fits is a slightly modified version of the same named official file already in CRDS, made unique by a small header change.
Cleanup: delete the rejected file from the upload area.

Case 3) - submitted file has a new name, unique contents, and auto-rename is deselected
1. Deselect auto-rename, and enter as the name of the file to submit ‘jwst_niriss_mask_0100.fits'.
2. Enter the file creator/author name and a text description.
3. Submit the file and confirm

Expected outcome: this file will be accepted because it has unique contents and a unique JWST-style name. The same name (0100) can only be used once. The same file contents (0100) can only be used once. (Subsequent tests on the same server will fail, unless the server is “reset” to initial conditions.)
Cleanup: none

Case 4) - submitted file has new name, same contents, auto-rename is irrelevant
1. Select or de-select auto rename.
2. Enter the file creator/author name and a text description.
3. Copy jwst_niriss_mask_0100.fits to jwst_niriss_mask_0101.fits and upload
4. Submit the file

Expected outcome: this file will be rejected based on contents identical to 0100.
Cleanup: delete the rejected file from the upload area.

9 Requirement ID: DMS-682

DMS Level 4 Requirement:
All CRDS rules and context files shall have unique names.

Verification Method:
Demonstration

Comments:

Verification Procedure:
This requirement can be verified in a manner similar to that of DMS-681. To verify this requirement, the tester must have an account on the CRDS website. For each case below, the tester should first login to CRDS and navigate to the Submit Mappings page. File naming for CRDS rules relies on a required canonical form. Rules files submitted to the web site can optionally be automatically renamed to a new serial number and/or standard form; files which are not renamed must be submitted in canonical form with a unique serial number. The CRDS server requires that rules files be submitted in dependency order, rmaps first, then imaps, then pmaps, all as independent submissions which have their lower level dependencies already in CRDS when they are submitted. The CRDS server does not automatically adjust the file names of inter-related files when auto-renaming is selected. Consequently, any file dependencies which are renamed must be resolved manually in higher level files (imaps or pmaps) prior to submitting the higher level files. There are two typical styles of file submission: (a) an entire rules network is submitted with renaming turned off, all base-lined on the same serial number. (b) a single level of rules (rmap, imap, or pmap) is downloaded, modified, and uploaded with the names unchanged but renaming selected. Typically these will be new or modified rmaps. CRDS automatically updates internal fields of the rules file tracking the new name in the header and moving the old name to the derived_from header field.

Case 1) – submitted file has a new name and unique contents, auto-rename is selected

1. Enter as the name of the file to submit 'my_map1.pmap'.
2. Submit the file.

Expected outcome: this should result in the file having a new name auto-generated, and the file accepted.

Case 2) - submitted file has a new name and unique contents, auto-rename is deselected

1. Deselect auto-rename, and enter as the name of the file to submit 'jwst_0400.pmap'.
2. Submit the file.

Expected outcome: this should result in the file having being accepted, with the requested name. It is not possible to submit names in arbitrary form, they must comply with the canonical form of CRDS rules names and by definition should use the latest available (or later) serial numbers.

Case 3) - submitted file has existing name, existing contents, auto-rename is deselected

3. Deselect auto-rename, and enter as the name of the file to submit 'jwst_0400.pmap'.
4. Submit the file.

Expected outcome: this should result in the file having being rejected since a same-named file already exists.

NOTE: “identical contents” checks are not supported for CRDS rules files since they are self-reflective and contain information about their current names and which rules they were derived from.