Nancy Grace Roman Space Telescope (Roman)

Technical Report

Title: Roman SOC Science Validation Report for SOC Release 1 DMS	Doc #: Roman-STScI-000456, SC-01 Date: December 1, 2022 Rev: -
Authors: Desjardins, T. D., Bellini, A., Cosentino, R. G., Hoffmann, S. L., and Sánchez, J. Phone: (667) 218-6457	Release Date: February 6, 2023

1 Executive Summary

This report contains the scientific validation of the following Science Operations Center (SOC) Data Management Subsystem (DMS) Level 4 systems engineering requirements for SOC Release 1. All requirements tested in this report have successfully passed scientific validation.

Table 1-1 Summary of SOC Level 4 Requirement Validation Results

SOC Level 4 Requirement ID	DMS Level 5 Requirement ID(s)	Level 4 Requirement	Validation Result [Pass/Fail]
SOC-303	DMS-16, DMS-18	The DMS shall make available for use WFI calibration reference data as needed to support WFI science data processing for Wide Field Imaging mode.	Pass

Operated by the Association of Universities for Research in Astronomy, Inc., for the National Aeronautics and Space Administration under Contract #80GSFC19C0054

SOC Level 4 Requirement ID	DMS Level 5 Requirement ID(s)	Level 4 Requirement	Validation Result [Pass/Fail]
SOC-587	DMS-129	In Wide Field Imaging mode, the DMS shall generate Level 2 science data products that incorporate information needed to correct for pixel geometric distortion.	Pass
SOC-588	DMS-140	In Wide Field Imaging mode, the DMS shall generate Level 2 science data products with absolute photometry calibrated in the WFI filter used for the exposure.	Pass
SOC-589	DMS-139	In Wide Field Imaging mode, the DMS shall generate Level 2 science data products with known relative zero-points of the filter photometry.	Pass
SOC-591	DMS-81	In Wide Field Imaging mode, the DMS shall generate Level 2 science data products with a relative photometric calibration in the WFI filter used for the exposure, compensated for spatial variation, over the Wide Field Channel field of view.	Pass
SOC-628	DMS-25, DMS-27	The DMS shall make available calibration reference data as needed to support WFI science data processing for Wide Field Spectroscopy mode observations.	Pass

2 Introduction

In November 2022, the Roman Telescope Branch (RTB) in the Instruments Division at the Space Telescope Science Institute (STScI) performed scientific validation of Element Level requirements levied on the Roman Science Operations Center (SOC) included in Release 1. The scientific validation includes two subsystems at the SOC: the Data Management Subsystem (DMS) and the Planning and Scheduling Subsystem (PSS). This report concerns the DMS science validation, while a companion report (Mutchler and De Rosa 2023) addresses the validation of the PSS. All six DMS requirements in Release 1 that were validated were related to

the creation of Level 2 science data products (i.e., science data that have had detector-level calibrations applied, but have not yet been combined in any way). The requirements involved the Level 2 science data pipeline functionality contained within the Exposure Level Processing component and the interfacing Calibration Reference Data System (CRDS). CRDS is used for version control and matching of reference files used during the calibration of science observations.

Science validation by the RTB differs from the verification process performed by Integration and Test in that the validation checks that specific inputs produce scientifically *correct* outputs that meet the user's expectations. Both verification and validation are important and necessary steps in assessing the successful development of the Roman DMS and documenting that development with the Roman Project.

SOC Release 1 included development work that was delivered for testing starting in SOC Build 0.0 up to and including SOC Build 22Q4_B7 (December 15, 2019 through September 15, 2022). Note the change in build naming that occurred during development of Release 1. In the previous system, SOC Build 22Q4_B7 would have been labeled as Build 0.7, whereas the new system includes the fiscal year and quarter in the build name with an incremental number. The new build nomenclature will be used exclusively in future validation reports, while we include this explanation here for backward compatibility with any documents that may concern early SOC builds.

To perform the validation, the individual testers were given a description of the steps necessary to test each requirement, a list of test procedures and guidelines, and computing environment specifications (see Appendix A). All validation code was run by the testers on the same virtual machine. Use of a common virtual machine and computing environment settings ensured that all tests were performed using identical environments.

In the Release 1 scientific validation, there were two pairs of requirements (SOC-303 and 628; and SOC-588 and 589) that were functionally the same with respect to the scientific validation; therefore, we group these requirements into these pairs for testing purposes. This yields a total of four validation tests that are described in the sections below. Appendices B through E reproduce the code used for the validation tests, and the Python files are also included as artifacts along with the test data files.

3 Scope of Testing

As previously stated, the scientific validation performed by the RTB checked that for a given input value, the correct outputs were generated, or that the correct algorithm was applied to the inputs. The RTB validation was performed against SOC Level 4 requirements that were assigned to the DMS. Note that not all Level 4 requirements in Release 1 required scientific validation, and that those included in this report are a subset of the DMS requirements that were identified for scientific validation by one or more of Roman SOC Systems Engineering, Integration and Test, DMS, and/or the RTB. Table 3-1 contains a list of the Level 4 requirements validated in this report, the requirement text, and the corresponding romancal step if applicable.

Table 3-1 SOC Level 4 Requirements for Release 1 Science Validation

SOC Level 4 Requirement ID	Level 4 Requirement	romancal Step	Notes
SOC-303	The DMS shall make available for use WFI calibration reference data as needed to support WFI science data processing for Wide Field Imaging mode.	All	Validated with the same method as SOC-628.
SOC-587	In Wide Field Imaging mode, the DMS shall generate Level 2 science data products that incorporate information needed to correct for pixel geometric distortion.	assign_wcs	
SOC-588	In Wide Field Imaging mode, the DMS shall generate Level 2 science data products with absolute photometry calibrated in the WFI filter used for the exposure.	photom	Validated with the same method as SOC-589.
SOC-589	In Wide Field Imaging mode, the DMS shall generate Level 2 science data products with known relative zero-points of the filter photometry.	photom	Validated with the same method as SOC-588.
SOC-591	In Wide Field Imaging mode, the DMS shall generate Level 2 science data products with a relative photometric calibration in the WFI filter used for the exposure, compensated for spatial variation, over the Wide Field Channel field of view.	flat_field	
SOC-628	The DMS shall make available calibration reference data as needed to support WFI science data processing for Wide Field Spectroscopy mode observations.	All	Validated with the same method as SOC-303.

4 Test Data

The input data used for testing consist of two types: 1) simulated science observations; and 2) test calibration reference files. The test data used for each requirement are a subset of these data and will be identified in sections below. Copies of all simulated science and calibration reference files will be made available to the Roman Project with this report.

Simulated Level 1 (i.e., uncalibrated ramp) science observation files were generated using WFIsim (Bellini, et al. 2022), which is a simulator written in Fortran by the RTB for testing the Roman SOC DMS. Four full-focal-plane images were generated with both point- and extended-sources using an input source catalog. For the validation of Release 1, the source

properties were not important, and we did not attempt to simulate dispersed sources (i.e., sources observed with the WFI grism or prism). At this level, the romancal steps do not care about the pixel data difference between imaging and dispersed observations, only the metadata inside the files that relates to logical switches within the romancal code. From the requirements being tested (see Section 3), the main goals of the Release 1 scientific validation are to test the ELP algorithms being used, and that correct calibration reference data are retrieved based on science observation file metadata. WFIsim outputs were in the Flexible Image Transport System (FITS) format (Wells, Greisen and Harten 1981), and were converted to Advanced Scientific Data Format (ASDF) standard (Greenfield, Droettboom and Bray 2015) to be compatible with the Roman SOC. The metadata of these ASDF files were manipulated to create the following:

1. A program containing a single visit consisting of a single Wide Field Instrument (WFI) Imaging Mode (WIM) exposure. The optical element of the exposure was set to F158 and the observation date was set to 2021-01-01T00:00:00 UTC.
2. A second program containing three visits each consisting of a single exposure:
    a. The first visit contains a WIM exposure in F158 with the date set to 2021-09-01T00:00:00 UTC.
    b. The second visit contains a WFI Spectroscopic Mode (WSM) exposure using the grism element and the date set to 2021-09-01T00:02:28.96 UTC.
    c. The third visit contains a WSM exposure using the prism element and the date set to 2021-09-01T00:04:57.92 UTC.

The breadth of these test data provides science observations that sample different points in time (and thus can test different calibration reference files based on date) as well as samples for both the WIM and WSM exposure types.

Note that both Level 1 and 2 files contain only one of the 18 WFI detectors. Thus, for each exposure described above, there are 18 files each for a total of 72 Level 1 test science observation files. The file names of the test data files are structured the same as Roman WFI science will be in operations, which is described by:

```
r{PPPPP}{XX}{ppp}{SSS}{OOO}{VVV}_{gg}{s}{aa}_{EEEE}_WFI{NN}_{uncal|cal}.asdf
```

where Table 4-1 describes each of values of in brackets. Table 4-2 gives a summary of the test science observation file description from the bullet points above.

Table 4-1 Roman Science Observation File Naming Convention

Abbreviation	Meaning
PPPPP	Program ID number
XX	Execution plan
ppp	Pass
SSS	Segment
OOO	Observation
VVV	Visit ID number
gg	Visit file group

Abbreviation	Meaning
s	Visit file sequence
aa	Visit file activity
EEEE	Exposure number
NN	WFI SCA number
uncal | cal	String that identifies the file level, i.e., uncal = Level 1 and cal = Level 2

Table 4-2 Test Science Observation File Specifications

File root name	Optical Element	Observation Datetime (UTC)
r0000101001001001001_01101_0001_WFI[01-18]	F158	2021-01-01T00:00:00.00
r0000201001001001001_01101_0001_WFI[01-18]	F158	2021-09-01T00:00:00.00
r0000201001001001002_01101_0001_WFI[01-18]	Grism	2021-09-01T00:02:28.96
r0000201001001001003_01101_0001_WFI[01-18]	Prism	2021-09-01T00:04:57.92

The calibration reference files used during scientific validation were stored on and retrieved from the Roman CRDS TEST server (https://roman-crds-test.stsci.edu), which can only be accessed within the STScI firewall at the time of writing. All testers used the same calibration reference files by setting the `CRDS_CONTEXT` environment variable to `roman_0039.pmap` (this is set automatically using the environment YAML file in Appendix A), which included an update to the date and time formatting in the reference files and is compatible with romancal version 0.8.1 according to regression testing performed by the DMS. The "pmap" file indicates which versions of calibration reference files should be used when querying CRDS. In this case, `roman_0039.pmap` was activated on August 23, 2022. All of the calibration reference files contained within this context conformed to the proper formats as defined in `romancal` documentation (see https://roman-pipeline.readthedocs.io/en/latest/) at the time RTB performed the scientific validation. In the future, previous versions of the romancal documentation can be built from the code if necessary. The calibration reference files were generated primarily using properties of the WFI detectors from the Phase C payload design that was baselined for the Fall 2021 Critical Design Review with additional Gaussian noise and appropriate features (e.g., hot pixels in the dark current reference files). Similar to the simulated science observations, the exact contents of the calibration reference file data arrays are not important for the validation testing in Release 1, simply that the calibration reference files are applied correctly by the ELP; therefore, we do not describe the calibration reference file contents in further detail in this report. The Science Instrument Aperture File (SIAF) was used to both generate the distortion reference file and also validate SOC-587. The SIAF was developed using simulated wavefront data from the Goddard Space Flight Center (GSFC) Optics Lab (Desjardins, et al. 2020). A copy of the Roman SIAF, which is configuration managed in the SOC Project Reference Database (PRD), will be made available to the Roman Project with this report.

5 Requirements Tested

5.1 SOC-303 and SOC-628: WFI Calibration Reference Data Availability

SOC-303 and SOC-628 require that the Exposure Level processing can retrieve calibration references files from the CRDS. SOC-303 refers to the imaging mode, while SOC-628 refers to the spectroscopic mode. For the validation of these requirements, we checked that the correct calibration reference files retrieved from the CRDS were used by the Exposure Level processing. These two requirements will be retested if additional calibration reference file types and calibration pipeline steps are added.

5.1.1 Calibration commands used

```
import os
import warnings

from romancal.pipeline.exposure_pipeline import ExposurePipeline

for file in files:

    # Check if files already exist and overwrite if necessary.
    # If file exists and overwrite = False, warn the user and
    # skip the calibration process.
    calibrate = True
    if os.path.exists(file.replace('uncal', 'cal')):
        if overwrite:
            os.remove(file.replace('uncal', 'cal'))
        else:
            warnings.warn(f"Skipping calibration. File
{file.replace('uncal', 'cal')} already exists and overwrite ==
{overwrite}!")
            calibrate = False

    if calibrate:
        args = ["--steps.jump.rejection_threshold=180.",
                "--steps.jump.three_group_rejection_threshold=20.",
                "--steps.jump.four_group_rejection_threshold=30.",
                "roman_elp",
                file]

        calpipe = ExposurePipeline()
        calpipe.from_cmdline(args)
```

5.1.2 Data used for testing

For this test, we used all of the test data files described in Section 4. This sample ensured that we adequately checked the different observing modes (WIM and WSM), the ability to have different reference files based on date and time of observation, and different reference files for each detector in the WFI.

5.1.3 Testing strategy

Exposure level processing creates output science data files that contain calibrated science data results. Each file also includes the set of metadata that describes the exposure within the file, along with processing log information. To test if the correct reference files were retrieved from CRDS, and used, for a given science exposure, the exposure logs within each file were examined and compared against recommendations from CRDS. The CRDS server located in the TEST string was queried using metadata from each science file with the CRDS context set to `roman_0039.pmap`. This query returned the name of the reference file that should be used with the corresponding science data file. These returned filenames were compared against the names recorded in the science data file logs. We also checked that the CRDS returned a value of "NOT FOUND n/a" for the flat field reference file for any input WSM observations. This is special behavior to indicate that the correct match is a non-match for this mode, rather than raising an Exception in Python. Python code for this test may be found in Appendix B.

5.1.4 Results

We found that the calibration files used matched the correct files on the CRDS TEST server for all test data files. We note that there are some inconsistencies in the formatting of the reference file information in the calibration logs of the science files in the Release 1 version of the Exposure Level pipeline. For example, the `assign_wcs` step writes the distortion reference file name in the log as part of a Python dictionary, while no other step uses that syntax. Other examples include inconsistent use of capilization (e.g., Linearity versus FLAT) and punctuation. These inconsistencies do not affect the outcome of the validation test; however, we urge the DMS developers to correct these in a future build of the Exposure Level pipeline. Jira ticket RCAL-432 was filed to track this log formatting bug.

5.2 SOC-587: Geometric Distortion Information for Wide Field Imaging

SOC-587 indicates that Level 2 WFI science data products should contain information necessary to correct for any geometric distortion present. This is accomplished during Exposure Level processing by embedding an Astropy (Astropy Collaboration et al. 2022) model into the science file metadata (stored as `meta.wcs`) as part of the `assign_wcs` step. The Astropy model contains the full transformation from the WFI science coordinate frame (Desjardins, et al. 2020) and the International Celestial Reference System (ICRS) sky coordinate frame. Part of the full transformation includes a conversion from the WFI science coordinate frame, which is distorted, to the distortion-corrected "ideal" coordinate frame, and subsequently the telescope (or V) frame (Desjardins, et al. 2020). The conversion from science to the ideal frame is contained within the distortion reference file retrieved from CRDS, while the remaining transformation to telescope and sky coordinates is added by `assign_wcs` and depends on the telescope pointing information contained within the science observation file metadata.

5.2.1 Calibration commands used

```
import os
import warnings

from romancal.pipeline.exposure_pipeline import ExposurePipeline

for file in files:

    # Check if files already exist and overwrite if necessary.
    # If file exists and overwrite = False, warn the user and
    # skip the calibration process.
    calibrate = True
    if os.path.exists(file.replace('uncal', 'cal')):
        if overwrite:
            os.remove(file.replace('uncal', 'cal'))
        else:
            warnings.warn(f"Skipping calibration. File
{file.replace('uncal', 'cal')} already exists and overwrite ==
{overwrite}!")
            calibrate = False

    if calibrate:
        args = ["--steps.jump.rejection_threshold=180.",
                "--steps.jump.three_group_rejection_threshold=20.",
                "--steps.jump.four_group_rejection_threshold=30.",
                "roman_elp",
                file]

        calpipe = ExposurePipeline()
        calpipe.from_cmdline(args)
```

5.2.2 Data used for testing

For this test, we used a single, complete exposure, i.e., all 18 WFI detectors for a given observation. As the choice of observation was not critically important and would not impact the outcome of the test, we chose the observation with the root file name r0000101001001001001_01101_0001, which was in the imaging mode with optical element F158. We note that at some later time there may be element-dependent distortions that must be corrected for, however this will be handled by selecting an element-dependent distortion reference file from the CRDS that contains the correct distortion model.

5.2.3 Testing strategy

To validate that the geometric distortion information was correctly added to the Level 2 science data products, we tested the full transformation from science to sky coordinates. If

successful, an end-to-end test of the full coordinate transformation asserts that the geometric distortion model has been correctly incorporated from the calibration reference file into the Level 2 science file.

We created a set of (x, y) science coordinate positions in a 32 x 32 evenly-spaced grid across each WFI detector. Note that Level 2 WFI products do not include the 4-pixel reference border or the additional 33rd amplifier readout, therefore they are 4088 x 4088 pixel arrays, and a grid spacing of 32 equates to 128 pixel spacing along each axis. At each grid position, we input the science coordinates into the `meta.wcs` model object to get the sky coordinates at that pixel position. Our truth values for comparison were determined using a Python package developed by STScI called `pysiaf` (Sahlmann, et al. 2022), which used the Roman SIAF (Desjardins, et al. 2020) in conjuction with an attitude matrix that incorporated telescope pointing information to do the transformation between different coordinate systems defined in the SIAF and the sky frame. The `pysiaf` code has been widely used for several years by the James Webb Space Telescope, and therefore we consider it to be robust. Python code for this test may be found in Appendix C.

5.2.4 Results

We found that the gwcs Python object inserted into the Level 2 WFI science files by the `assign_wcs` step during the Exposure Level processing correctly translated science pixel positions into sky coordinates. Comparison between the gwcs object outputs and the sky coordinates translated by the `pysiaf` package agreed with the gwcs sky coordinates within a floating point error tolerance of $< 1 \times 10^{-7}$ arcseconds.

5.3 SOC-588 and SOC-589: Absolute Photometric Calibration and Photometric Calibration Zero-Points

SOC-588 and SOC-589 are functionally the same in that they both concern the need to populate information into the Level 2 science data products that facilitate the conversion from instrumental units of electrons per second ($e^- s^{-1}$) into physical units. In the case of *Roman*, carrying over from the *James Webb Space Telescope* model, these physical units are megaJanskys per steradian (MJy sr^{-1}) and microJanskys per square arcsecond (μJy $arcsec^{-2}$), with conversion information provided in the Level 2 file metadata for both unit systems. The conversion also requires information about the sizes of the pixels on the sky, which is supplied as a nominal pixel area in both steradians and square arcseconds in the metadata.

While there are science requirements that detail the level of the photometric accuracy needed for *Roman*, we do not check the accuracy here. The photometric accuracy achieved in the Level 2 WFI science files depends on the contents of the photometric calibration reference file, which at the time of writing contains estimates from modeling. Further ground testing and on-orbit calibration observations will ensure that the accuracy requirements are met. For validation purposes, we checked that the Exposure Level processing correctly populated the information obtained from the calibration reference files into the Level 2 WFI science file metadata.

5.3.1 Calibration commands used

```
import os
```

```
import warnings

from romancal.pipeline.exposure_pipeline import ExposurePipeline

for file in files:

    # Check if files already exist and overwrite if necessary.
    # If file exists and overwrite = False, warn the user and
    # skip the calibration process.
    calibrate = True
    if os.path.exists(file.replace('uncal', 'cal')):
        if overwrite:
            os.remove(file.replace('uncal', 'cal'))
        else:
            warnings.warn(f"Skipping calibration. File
{file.replace('uncal', 'cal')} already exists and overwrite ==
{overwrite}!")
            calibrate = False

    if calibrate:
        args = ["--steps.jump.rejection_threshold=180.",
                "--steps.jump.three_group_rejection_threshold=20.",
                "--steps.jump.four_group_rejection_threshold=30.",
                "roman_elp",
                file]

        calpipe = ExposurePipeline()
        calpipe.from_cmdline(args)
```

5.3.2 Data used for testing

For this test, we used only one WFI detector (WFI05) with each of the F158, GRISM, and PRISM optical elements. We did not need to check different observation dates, so only one F158 image was used during the test. The following three files used were r0000101001001001001_01101_0001_WFI05_uncal.asdf, r0000201001001001002_01101_0001_WFI05_uncal.asdf, and r0000201001001001003_01101_0001_WFI05_uncal.asdf, which correspond to the F158, GRISM, and PRISM optical elements, respectively.

5.3.3 Testing strategy

Level 1 science files were processed through the Exposure Level processing, and the metadata values in the `meta.photometry` section were checked to be correctly populated with photometry information. For WIM files, these metadata values should be Astropy quantity objects with both the correct value and units, whereas for WSM files all the photometric

conversion keywords should be set to Python None by agreement with the Science Support Center (SSC). The photom reference files contain only the conversion to MJy sr^{-1}, and calibration processing converts these values to μJy $arcsec^{-2}$. We checked this conversion between physical unit systems as well. Python code used for the test may be found in Appendix D.

5.3.4 Results

We found that the photometric keywords in the Level 2 WIM science file metadata were correctly populated with the required information. For the WSM case, the Level 2 file contained values of Python None as expected. We also checked the conversion between the MJy sr^{-1} and μJy $arcsec^{-2}$ unit systems and found that these values agree with our expectations. Thus, the photometric calibration step in Exposure Level processing correctly implements the requirements described in SOC-588 and 589.

5.4 SOC-591: Relative Photometric Calibration

While SOC-591 refers to the relative photometric calibration of WIM data, the text of the Level 4 requirement further describes that spatial variation in the photometric calibration should be corrected. A comment on the requirement elaborates that this is interpreted as the application of the flat-field correction to the data. While the Science Requirements Document (SRD) states a specific accuracy (i.e., less than 1% RMS) that this correction should be accomplished within, that accuracy is limited by the quality of the calibration reference file. Thus, we do not focus here on the accuracy as described in the SRD, but rather on the correct application of the contents of the flat-field calibration reference file to the science data.

5.4.1 Calibration commands used

```
import os
import warnings

from romancal.dark_current.dark_current_step import DarkCurrentStep
from romancal.dq_init.dq_init_step import DQInitStep
from romancal.flatfield.flat_field_step import FlatFieldStep
from romancal.jump.jump_step import JumpStep
from romancal.linearity.linearity_step import LinearityStep
from romancal.ramp_fitting.ramp_fit_step import RampFitStep
from romancal.saturation.saturation_step import SaturationStep
from romancal.assign_wcs.assign_wcs_step import AssignWcsStep

for file in files:

    # Check if files already exist and overwrite if necessary.
    # If file exists and overwrite = False, warn the user and
    # skip the calibration process.
    calibrate = True
    if os.path.exists(file.replace('uncal', 'assignwc')):
```

```
        if overwrite:
            os.remove(file.replace('uncal', 'assignwcs'))
        else:
            warnings.warn(f"Skipping calibration. File
{file.replace('uncal', 'assignwcs')} already exists and overwrite ==
{overwrite}!")
            calibrate = False
    if os.path.exists(file.replace('uncal', 'flat')):
        if overwrite:
            os.remove(file.replace('uncal', 'flat'))
        else:
            warnings.warn(f"Skipping calibration. File
{file.replace('uncal', 'flat')} already exists and overwrite ==
{overwrite}!'")
            calibrate = False

    if calibrate:
        input_data = rdm.open(file)

        result = DQInitStep.call(input_data)
        result = SaturationStep.call(result)
        result = DarkCurrentStep.call(result)
        result = LinearityStep.call(result)
        result = JumpStep.call(result, rejection_threshold=180.,
                               three_group_rejection_threshold=20.,
                               four_group_rejection_threshold=30.)
        result = RampFitStep.call(result)
        result = AssignWcsStep.call(result, save_results=True)
        result = FlatFieldStep.call(result, save_results=True)
```

5.4.2 Data used for testing

Each detector is calibrated separately, and the correction only depends on the specification of detector and optical element in the metadata. Thus, to sufficiently validate the flat-field correction step, we used one detector (WFI05) and three optical elements: F158; GRISM; and PRISM. These three files are r0000101001001001001_01101_0001_WFI05_uncal.asdf, r0000201001001001002_01101_0001_WFI05_uncal.asdf, and r0000201001001001003_01101_0001_WFI05_uncal.asdf, which correspond to the F158, GRISM, and PRISM optical elements, respectively. The test files contain pixel data with identical information, and only the metadata are different to perform this test.

5.4.3 Testing strategy

The correct application of the flat-field calibration reference file is to divide the science data array by the flat-field array. We validated the flat-field step by computing the ratio of the science data array input into the flat-field step (i.e., the output of the previously run assign_wcs

step) and the output array of the flat-field step. For WIM data, this ratio should equal the flat-field array. While SOC-591 applies strictly to WIM data, for WSM data the ratio should be unity. This latter case is because no flat-field correction should be applied to WSM observations. At the time of this report, the SSC plans in operations to flat-field correct the WSM observations as part of their WFI spectroscopic pipeline. We checked both of the WIM and WSM cases for our science validation of SOC-591.

5.4.4 Results

For WIM science data, we found that the ratio of the flat-field step input and output arrays equaled the flat-field reference file for all but 7 pixels. Further investigation of these 7 pixels showed that the ratio array contained a value of NaN, and that the data quality (DQ) arrays for those pixels had values of 3. DQ arrays are bitwise additions of up to 32 bits that correspond to different information about the pixel status in the ELP. A value of zero in the DQ array indicates that there were no problems during the calibration process. A value of 3 is the combination of bits 0 and 1 (i.e., $2^0 = 1$ and $2^1 = 2$), which correspond to "DO NOT USE" and "SATURATED." In this case, those DQ bits mean that these pixels are saturated in every resultant in the exposure, and thus the output from the ramp-fitting step is set to 0 $e^- s^{-1}$ in the affected pixels. When we computed the ratio of the input and output arrays of the flat-field step, we divided 0 by 0 in NumPy arrays, and the result was a NaN.

For WSM science data, we found that the input and output arrays of the flat-field step were identical, which was expected for the WSM data.

For both the WIM and WSM science data, we find that the flat-field step successfully implements the spatial variation of the relative photometric calibration as described in requirement SOC-591.

6 Conclusions

We have designed and run scientific validation tests on six SOC Level 4 requirements for the Roman DMS as part of SOC Release 1. The requirement numbers and the result of their validation tests are shown below in Table 6-1. All requirements that have undergone scientific validation in Release 1 have received a pass.

Table 6-1 Summary of Validation Results

SOC Level 4 Requirement ID	Level 4 Requirement	Validation Result [Pass/Fail]
SOC-303	The DMS shall make available for use WFI calibration reference data as needed to support WFI science data processing for Wide Field Imaging mode.	Pass
SOC-587	In Wide Field Imaging mode, the DMS shall generate Level 2 science data products that incorporate information needed to correct for pixel geometric distortion.	Pass

SOC Level 4 Requirement ID	Level 4 Requirement	Validation Result [Pass/Fail]
SOC-588	In Wide Field Imaging mode, the DMS shall generate Level 2 science data products with absolute photometry calibrated in the WFI filter used for the exposure.	Pass
SOC-589	In Wide Field Imaging mode, the DMS shall generate Level 2 science data products with known relative zero-points of the filter photometry.	Pass
SOC-591	In Wide Field Imaging mode, the DMS shall generate Level 2 science data products with a relative photometric calibration in the WFI filter used for the exposure, compensated for spatial variation, over the Wide Field Channel field of view.	Pass
SOC-628	The DMS shall make available calibration reference data as needed to support WFI science data processing for Wide Field Spectroscopy mode observations.	Pass

7 Bibliography

Astropy Collaboration et al. 2022. "The Astropy Project: Sustaining and Growing a Community-oriented Open-source Project and the Latest Major Release (v5.0) of the Core Package." *ApJ* 935, 167.

Bellini, A., T. D. Desjardins, S. Casertano, and R. E. Ryan. 2022. "WFIsim: The Roman Telescope Branch Wide-Field-Instrument Simulator." Roman-STScI-000433.

Desjardins, T., R. Ryan, M. Sosey, S. Casertano, and C.-P. Lajoie. 2020. "Description of the Roman SIAF and Coordinate Frames." Roman-STScI-000143 Rev A.

Greenfield, P., M. Droettboom, and E. Bray. 2015. "ASDF: A New Data Format for Astronomy." *Astronomy and Computing* 12: 240-251.

Mutchler, M., and G. De Rosa. 2023. "Roman SOC Science Validation Report for SOC Build 8 Planning & Scheduling System (PSS) Build 1.0." Roman-STScI-000457.

Sahlmann, J., S. Osborne, M. Fix, D. Long, C. Cox, D. Law, charlesrp, et al. 2022. https://zenodo.org/record/6800239.

Wells, D. C., E. W. Greisen, and R. H. Harten. 1981. "FITS – A Flexible Image Transport System." *A&AS* 44, 363.

Appendix A. System and Computing Environment Specifications

The SOC DMS Release 1 science validation was performed on a Linux virtual machine in the STScI Flexible Data Center. The machine (dlrrtb1.stsci.edu) uses 18 Intel® Xeon® Gold 6254 CPUs with clock speeds of 3.10 GHz, and 64 GB of DDR-4 2933 MHz RAM. The operating system during testing was Red Hat Enterprise Linux (RHEL) 8.6 with Linux kernel 4.18.0-372.32.1.el8_6.x86_64.

Conda version 22.9.0 was used to control the computing environment for the science validation. An environment YAML file (reproduced below) was created and shared with the testers to make sure that all versions of software dependencies were the same as well as the CRDS context that controls the calibration reference file versioning. A copy of the YAML file is also included with the validation test artifacts.

The romancal package and its dependencies compose the implementation of the Roman science calibration pipeline. The version of romancal tested in this report is version 0.8.1 (https://github.com/spacetelescope/romancal/releases/tag/0.8.1), and this release was delivered to support functionality through SOC/DMS Build 22Q4_B7.

```
name: rdms_r1_val
channels:
  - conda-forge
  - defaults
dependencies:
  - _libgcc_mutex=0.1=conda_forge
  - _openmp_mutex=4.5=2_gnu
  - alsa-lib=1.2.7.2=h166bdaf_0
  - argon2-cffi=21.3.0=pyhd8ed1ab_0
  - argon2-cffi-bindings=21.2.0=py39hb9d737c_2
  - asttokens=2.0.8=pyhd8ed1ab_0
  - attr=2.5.1=h166bdaf_1
  - attrs=22.1.0=pyh71513ae_1
  - backcall=0.2.0=pyh9f0ad1d_0
  - backports=1.0=py_2
  - backports.functools_lru_cache=1.6.4=pyhd8ed1ab_0
  - beautifulsoup4=4.11.1=pyha770c72_0
  - bleach=5.0.1=pyhd8ed1ab_0
  - brotli=1.0.9=h166bdaf_7
  - brotli-bin=1.0.9=h166bdaf_7
  - bzip2=1.0.8=h7f98852_4
  - ca-certificates=2022.9.24=ha878542_0
  - certifi=2022.9.24=pyhd8ed1ab_0
  - cffi=1.15.1=py39he91dace_0
  - contourpy=1.0.5=py39hf939315_0
```

```
- cycler=0.11.0=pyhd8ed1ab_0
- dbus=1.13.6=h5008d03_3
- debugpy=1.6.3=py39h5a03fae_0
- decorator=5.1.1=pyhd8ed1ab_0
- defusedxml=0.7.1=pyhd8ed1ab_0
- entrypoints=0.4=pyhd8ed1ab_0
- executing=1.1.1=pyhd8ed1ab_0
- expat=2.4.9=h27087fc_0
- fftw=3.3.10=nompi_hf0379b8_105
- flit-core=3.7.1=pyhd8ed1ab_0
- font-ttf-dejavu-sans-mono=2.37=hab24e00_0
- font-ttf-inconsolata=3.000=h77eed37_0
- font-ttf-source-code-pro=2.038=h77eed37_0
- font-ttf-ubuntu=0.83=hab24e00_0
- fontconfig=2.14.0=hc2a2eb6_1
- fonts-conda-ecosystem=1=0
- fonts-conda-forge=1=0
- fonttools=4.37.4=py39hb9d737c_0
- freetype=2.12.1=hca18f0e_0
- gettext=0.19.8.1=h27087fc_1009
- glib=2.74.0=h6239696_0
- glib-tools=2.74.0=h6239696_0
- gst-plugins-base=1.20.3=h57caac4_2
- gstreamer=1.20.3=hd4edc92_2
- icu=70.1=h27087fc_0
- importlib-metadata=4.11.4=py39hf3d152e_0
- importlib_resources=5.10.0=pyhd8ed1ab_0
- ipykernel=6.16.0=pyh210e3f2_0
- ipython=8.5.0=pyh41d4057_1
- ipython_genutils=0.2.0=py_1
- ipywidgets=8.0.2=pyhd8ed1ab_1
- jack=1.9.18=h8c3723f_1003
- jedi=0.18.1=pyhd8ed1ab_2
- jinja2=3.1.2=pyhd8ed1ab_1
- jpeg=9e=h166bdaf_2
- jupyter=1.0.0=py39hf3d152e_7
- jupyter_client=7.4.2=pyhd8ed1ab_0
- jupyter_console=6.4.4=pyhd8ed1ab_0
- jupyter_core=4.11.1=py39hf3d152e_0
- jupyterlab_pygments=0.2.2=pyhd8ed1ab_0
- jupyterlab_widgets=3.0.3=pyhd8ed1ab_0
- keyutils=1.6.1=h166bdaf_0
```

```
- kiwisolver=1.4.4=py39hf939315_0
- krb5=1.19.3=h3790be6_0
- lcms2=2.12=hddcbb42_0
- ld_impl_linux-64=2.36.1=hea4e1c9_2
- lerc=4.0.0=h27087fc_0
- libblas=3.9.0=16_linux64_openblas
- libbrotlicommon=1.0.9=h166bdaf_7
- libbrotlidec=1.0.9=h166bdaf_7
- libbrotlienc=1.0.9=h166bdaf_7
- libcap=2.65=ha37c62d_0
- libcblas=3.9.0=16_linux64_openblas
- libclang=14.0.6=default_h2e3cab8_0
- libclang13=14.0.6=default_h3a83d3e_0
- libcups=2.3.3=h3e49a29_2
- libdb=6.2.32=h9c3ff4c_0
- libdeflate=1.14=h166bdaf_0
- libedit=3.1.20191231=he28a2e2_2
- libevent=2.1.10=h9b69904_4
- libffi=3.4.2=h7f98852_5
- libflac=1.3.4=h27087fc_0
- libgcc-ng=12.1.0=h8d9b700_16
- libgfortran-ng=12.1.0=h69a702a_16
- libgfortran5=12.1.0=hdcd56e2_16
- libglib=2.74.0=h7a41b64_0
- libgomp=12.1.0=h8d9b700_16
- libiconv=1.17=h166bdaf_0
- liblapack=3.9.0=16_linux64_openblas
- libllvm14=14.0.6=he0ac6c6_0
- libnsl=2.0.0=h7f98852_0
- libogg=1.3.4=h7f98852_1
- libopenblas=0.3.21=pthreads_h78a6416_3
- libopus=1.3.1=h7f98852_1
- libpng=1.6.38=h753d276_0
- libpq=14.5=hd77ab85_0
- libsndfile=1.0.31=h9c3ff4c_1
- libsodium=1.0.18=h36c2ea0_1
- libsqlite=3.39.4=h753d276_0
- libstdcxx-ng=12.1.0=ha89aaad_16
- libtiff=4.4.0=h55922b4_4
- libtool=2.4.6=h9c3ff4c_1008
- libudev1=249=h166bdaf_4
- libuuid=2.32.1=h7f98852_1000
```

```
- libvorbis=1.3.7=h9c3ff4c_0
- libwebp-base=1.2.4=h166bdaf_0
- libxcb=1.13=h7f98852_1004
- libxkbcommon=1.0.3=he3ba5ed_0
- libxml2=2.10.2=h7463322_2
- libzlib=1.2.12=h166bdaf_4
- markupsafe=2.1.1=py39hb9d737c_1
- matplotlib=3.6.1=py39hf3d152e_0
- matplotlib-base=3.6.1=py39hf9fd14e_0
- matplotlib-inline=0.1.6=pyhd8ed1ab_0
- mistune=2.0.4=pyhd8ed1ab_0
- munkres=1.1.4=pyh9f0ad1d_0
- mysql-common=8.0.31=haf5c9bc_0
- mysql-libs=8.0.31=h28c427c_0
- nbclient=0.7.0=pyhd8ed1ab_0
- nbconvert=7.2.1=pyhd8ed1ab_0
- nbconvert-core=7.2.1=pyhd8ed1ab_0
- nbconvert-pandoc=7.2.1=pyhd8ed1ab_0
- nbformat=5.7.0=pyhd8ed1ab_0
- ncurses=6.3=h27087fc_1
- nest-asyncio=1.5.6=pyhd8ed1ab_0
- notebook=6.4.12=pyha770c72_0
- nspr=4.32=h9c3ff4c_1
- nss=3.78=h2350873_0
- numpy=1.23.3=py39hba7629e_0
- openjpeg=2.5.0=h7d73246_1
- openssl=1.1.1q=h166bdaf_0
- packaging=21.3=pyhd8ed1ab_0
- pandoc=2.19.2=ha770c72_0
- pandocfilters=1.5.0=pyhd8ed1ab_0
- parso=0.8.3=pyhd8ed1ab_0
- pcre2=10.37=hc3806b6_1
- pexpect=4.8.0=pyh9f0ad1d_2
- pickleshare=0.7.5=py_1003
- pillow=9.2.0=py39hd5dbb17_2
- pip=22.2.2=pyhd8ed1ab_0
- pkgutil-resolve-name=1.3.10=pyhd8ed1ab_0
- ply=3.11=py_1
- portaudio=19.6.0=h8e90077_6
- prometheus_client=0.14.1=pyhd8ed1ab_0
- prompt-toolkit=3.0.31=pyha770c72_0
- prompt_toolkit=3.0.31=hd8ed1ab_0
```

```
- psutil=5.9.2=py39hb9d737c_0
- pthread-stubs=0.4=h36c2ea0_1001
- ptyprocess=0.7.0=pyhd3deb0d_0
- pulseaudio=14.0=h0868958_9
- pure_eval=0.2.2=pyhd8ed1ab_0
- pycparser=2.21=pyhd8ed1ab_0
- pygments=2.13.0=pyhd8ed1ab_0
- pyparsing=3.0.9=pyhd8ed1ab_0
- pyqt=5.15.7=py39h18e9c17_0
- pyqt5-sip=12.11.0=py39h5a03fae_0
- pyrsistent=0.18.1=py39hb9d737c_1
- python=3.9.13=h9a8a25e_0_cpython
- python-dateutil=2.8.2=pyhd8ed1ab_0
- python-fastjsonschema=2.16.2=pyhd8ed1ab_0
- python_abi=3.9=2_cp39
- pyzmq=24.0.1=py39headdf64_0
- qt-main=5.15.6=hc525480_0
- qtconsole=5.3.2=pyhd8ed1ab_0
- qtconsole-base=5.3.2=pyha770c72_0
- qtpy=2.2.1=pyhd8ed1ab_0
- readline=8.1.2=h0f457ee_0
- send2trash=1.8.0=pyhd8ed1ab_0
- setuptools=65.4.1=pyhd8ed1ab_0
- sip=6.6.2=py39h5a03fae_0
- six=1.16.0=pyh6c4a22f_0
- soupsieve=2.3.2.post1=pyhd8ed1ab_0
- sqlite=3.39.4=h4ff8645_0
- stack_data=0.5.1=pyhd8ed1ab_0
- terminado=0.16.0=pyh41d4057_0
- tinycss2=1.1.1=pyhd8ed1ab_0
- tk=8.6.12=h27826a3_0
- toml=0.10.2=pyhd8ed1ab_0
- tornado=6.2=py39hb9d737c_0
- traitlets=5.4.0=pyhd8ed1ab_0
- typing_extensions=4.4.0=pyha770c72_0
- tzdata=2022e=h191b570_0
- unicodedata2=14.0.0=py39hb9d737c_1
- wcwidth=0.2.5=pyh9f0ad1d_2
- webencodings=0.5.1=py_1
- wheel=0.37.1=pyhd8ed1ab_0
- widgetsnbextension=4.0.3=pyhd8ed1ab_0
- xcb-util=0.4.0=h166bdaf_0
```

```
- xcb-util-image=0.4.0=h166bdaf_0
- xcb-util-keysyms=0.4.0=h166bdaf_0
- xcb-util-renderutil=0.3.9=h166bdaf_0
- xcb-util-wm=0.4.1=h166bdaf_0
- xorg-libxau=1.0.9=h7f98852_0
- xorg-libxdmcp=1.1.3=h7f98852_0
- xz=5.2.6=h166bdaf_0
- zeromq=4.3.4=h9c3ff4c_1
- zipp=3.9.0=pyhd8ed1ab_0
- zstd=1.5.2=h6239696_4
- pip:
  - alabaster==0.7.12
  - asdf==2.13.0
  - asdf-astropy==0.2.2
  - asdf-coordinates-schemas==0.1.0
  - asdf-standard==1.0.3
  - asdf-transform-schemas==0.3.0
  - asdf-wcs-schemas==0.1.1
  - astropy==5.1
  - babel==2.10.3
  - charset-normalizer==2.1.1
  - crds==11.16.14
  - docutils==0.19
  - et-xmlfile==1.1.0
  - filelock==3.8.0
  - gwcs==0.18.2
  - idna==3.4
  - imagesize==1.4.1
  - jmespath==1.0.1
  - jsonschema==4.9.1
  - lxml==4.9.1
  - numpydoc==1.5.0
  - opencv-python==4.6.0.66
  - openpyxl==3.0.10
  - parsley==1.3
  - pyerfa==2.0.0.1
  - pysiaf==0.18.0
  - pytz==2022.4
  - pyyaml==6.0
  - rad==0.13.2
  - requests==2.28.1
  - roman-datamodels==0.13.0
```

```
    - romancal==0.8.1
    - scipy==1.9.2
    - semantic-version==2.10.0
    - snowballstemmer==2.2.0
    - soc-roman-tools==0.1.0
    - sphinx==5.2.3
    - sphinxcontrib-applehelp==1.0.2
    - sphinxcontrib-devhelp==1.0.2
    - sphinxcontrib-htmlhelp==2.0.0
    - sphinxcontrib-jsmath==1.0.1
    - sphinxcontrib-qthelp==1.0.3
    - sphinxcontrib-serializinghtml==1.1.5
    - stcal==1.2.0
    - stdatamodels==0.4.3
    - stpipe==0.4.2
    - urllib3==1.26.12
variables:
  CRDS_CONTEXT: roman_0039.pmap
  CRDS_PATH: ./cache
  CRDS_SERVER_URL: https://roman-crds-test.stsci.edu
```

Appendix B. SOC-303 and SOC-628 Validation Code

```
from crds import getrecommendations
import roman_datamodels as rdm

def prRed(s):
    """
    Simple function to print messages to stdout in red.

    Inputs
    ------
    s (str): String to print.

    Returns
    -------
    None
    """
    print(f"\033[91m {s}\033[00m")

def main(file):
    """
    Code to do the validation test. See science validation report for
    Test description.

    Inputs
    ------
    file (str): Name of the Level 2 file to be validated.

    Returns
    -------
    None
    """

    print(f'\nWorking on file {file}...')

    # Get the calibration pipeline logs and the meta data out of the
    # ASDF file.
    data = rdm.open(file)
    meta = data.meta
    logs = data.cal_logs

    # Pull the names of the reference files that were used out of the
```

```
# calibration logs.
log_files = {}
for line in logs:
     sstr = './cache/references/roman/wfi/'
    if './cache/references' in line:
        if 'assign_wcs' not in line:
            ref_type = line.split(':: Using ')[1].split()[0].lower()
            ref_file = line.split(sstr)[1].rstrip()
        else:
            ref_type = 'distortion'
            ref_file = line.split(sstr)[1][:-2].rstrip()
        log_files.update({ref_type: ref_file})

# Construct the selector information for CRDS.
selectors = {'ROMAN.META.INSTRUMENT.NAME': 'WFI',
             'ROMAN.META.INSTRUMENT.DETECTOR':
                            meta.instrument.detector,
             'ROMAN.META.INSTRUMENT.OPTICAL_ELEMENT':
                            meta.instrument.optical_element,
             'ROMAN.META.EXPOSURE.START_TIME':
                            meta.exposure.start_time.isot,
             'ROMAN.META.EXPOSURE.MA_TABLE_NUMBER':
                            meta.exposure.ma_table_number,
             'ROMAN.META.EXPOSURE.TYPE': meta.exposure.type}

# Get the correct reference file names from the CRDS server.
reftypes = list(log_files.keys())
if 'flat' not in reftypes:
    reftypes.append('flat')
crds_files = getrecommendations(selectors, reftypes=reftypes,
                                observatory='roman',
                                context='roman_0039.pmap')

# Test if the CRDS returned files match what was used by romancal.
# For the flats, if the data were in WSM, we expect
# 'not found n/a' to appear in the CRDS results.
all_tests = []
for key, val in crds_files.items():
    if (key == 'flat') and (meta.exposure.type != 'WFI_IMAGE'):
        test = val.lower() == 'not found n/a'
    else:
        test = val == log_files[key]
```

```
    all_tests.append(test)

if False in all_tests:
    prRed('\tTEST FAILED!')
else:
    print('\tTEST PASSED!')
```

Appendix C. SOC-587 Validation Code

```
import numpy as np

from pysiaf.utils.rotations import attitude, pointing
import roman_datamodels as rdm
from soc_roman_tools.siaf.siaf import RomanSiaf

def prRed(s):
    """
    Simple function to print messages to stdout in red.

    Inputs
    ------
    s (str): String to print.

    Returns
    -------
    None
    """
    print(f"\033[91m {s}\033[00m")

def main(file):
    """
    Code to do the validation test. See science validation report for
    test description.

    Inputs
    ------
    file (str): Name of the Level 2 file to be validated.

    Returns
    -------
    None
    """

    # Made grid of science coordinate (x, y) positions in a 32 x 32
    # evenly-spaced grid. Note that the gwcs object assumes 0-indexed
    # Python pixel positions, while the SIAF will assume 1-indexed
    # pixel positions.
    sci_x = np.arange(0, 4087, 128)
    sci_y = sci_x.copy()
```

```
    sci_xg, sci_yg = np.meshgrid(sci_x, sci_y)
    sci_xx = sci_xg.flatten()
    sci_yy = sci_yg.flatten()

    print(f"\nWorking on science file rootname =
{f.split('_cal.asdf')[0]}...")

    # Open the ASDF file and get the detector ID, the gwcs object,
    # the ra, dec, and roll angle at the reference pixel position
    # on the detector.
    meta = rdm.open(f).meta
    det_id = meta.instrument.detector
    wcs = meta.wcs
    ra_ref = meta.wcsinfo.ra_ref
    dec_ref = meta.wcsinfo.dec_ref
    roll_ref = meta.wcsinfo.roll_ref

    # Get the sky positions according to the gwcs object
    # in the Level 2 file.
    ra_wcs, dec_wcs = wcs(sci_xx, sci_yy)

    # Read the Roman SIAF and select the correct aperture.
    siaf = RomanSiaf()
    aper = siaf[f'{det_id}_FULL']

    # Construct an attiude matrix to use with pysiaf
    att = attitude(aper.V2Ref, aper.V3Ref, ra_ref, dec_ref, roll_ref)

    # Convert from science to telescope coordinates using the SIAF
    # remembering that the pixels need to be 1-indexed.
    v2, v3 = aper.sci_to_tel(sci_xx + 1, sci_yy + 1)
    ra_siaf, dec_siaf = pointing(att, v2, v3)

    # Test if the positions agree or not. Agreement should be
    # < 1e-7 arcsec to accommodate floating point error.
    ra_check = np.isclose(ra_wcs, ra_siaf)
    dec_check = np.isclose(dec_wcs, dec_siaf)

    # Check for pass or fail.
    if (ra_check.all()) and (dec_check.all()):
        print('\tPass!')
    else:
        prRed('\tFail!')
```

Appendix D. SOC-588 and SOC-589 Validation Code

```
import astropy.units as u
import roman_datamodels as rdm

def main(file):
    """
    Code to do the validation test. See science validation report for
    test description.

    Inputs
    ------
    files (list): A list of Level 2 file names to test.

    Returns
    -------
    None
    """

    meta = rdm.open(file).meta
    phot = meta.photometry
    print(f'\nFile = {file}\nMode = {meta.exposure.type}')
    print(f"{'-' * 16}\n| TYPE TESTING |\n{'-' * 16}")
    for key, val in phot.items():
        # Check if photometry keywords are astropy.units.Quantity
        # objects in WIM, else Python NoneType in WSM data.
        if meta.exposure.type == 'WFI_IMAGE':
            type_test = isinstance(val, u.Quantity)
        else:
            type_test = not val
        print(f'{key} = {type_test}')

    if meta.exposure.type == 'WFI_IMAGE':
        # For WIM data, check that photometry keywords have the
        # correct units.
        print(f"{'-' * 16}\n| UNIT TESTING |\n{'-' * 16}")
        for key, val in phot.items():
            if 'mega' in key:
                unit_test = u.Unit('MJy / sr')
            elif 'micro' in key:
                unit_test = u.Unit('uJy / arcsec^2')
            elif 'steradian' in key:
```

```
        unit_test = u.Unit('sr')
    else:
        unit_test = u.Unit('arcsec^2')
    print(f'{key} ({unit_test}) = {unit_test == val.unit}')

# For WIM data, check that the values of the photometry
# keywords in different units are equivalent to each
# other.
print(f"{'-' * 19}\n| CONVERT TESTING |\n{'-' * 19}")
for key, val in phot.items():
    if key == 'conversion_microjanskys':
        comp_val = phot.conversion_megajanskys
        value_test = u.isclose(val, comp_val)
        print(f'{key} value check = {value_test}')
    elif key == 'pixelarea_arcsecsq':
        comp_val = phot.pixelarea_steradians
        value_test = u.isclose(val, comp_val)
        print(f'{key} value check = {value_test}')
    elif key == 'conversion_microjanskys_uncertainty':
        comp_val = phot.conversion_megajanskys_uncertainty
        value_test = u.isclose(val, comp_val)
        print(f'{key} value check = {value_test}')
    else:
        pass
```

Appendix E. SOC-591 Validation Code

```
import numpy as np

import roman_datamodels as rdm

def prRed(s):
    """
    Simple function to print messages to stdout in red.

    Inputs
    ------
    s (str): String to print.

    Returns
    -------
    None
    """
    print(f"\033[91m {s}\033[00m")

def get_bits(x):
    """
    Function to decompose DQ values into the individual bits.
    """

    powers = []
    i = 1
    while i <= x:
        if i & x:
            powers.append(i)
        i <<= 1
    return powers

def main(wcsstep, flatstep, flat_reference=None):
    """
    Code to do the validation test. See science validation report for
     test
    description.

    Inputs
    ------
    wcsstep (str): File output from assign_wcs.
```

```
flatstep (str): File output from flatfield.

flat_reference (str; default=None): Name of the flat field reference
  file. Default is None.
     If wcsstep and flatstep are imaging mode data, flat_reference
  cannot be None or an
     exception will be raised.

Returns
-------
None
"""

print(f"\nWorking     on     science     file     rootname     =
  {wcsstep.split('_assignwcs.asdf')[0]}...")

if flat_reference:
    try:
        flat_ref = rdm.open(flat_reference)
    except:
        flat_ref = None
else:
    flat_ref = None

wfile = rdm.open(wcsstep)
ffile = rdm.open(flatstep)

wim = False
if wfile.meta.exposure.type == 'WFI_IMAGE':
    wim = True
    if not flat_ref:
        raise ValueError('If checking WIM data, you need a valid
  flat field refernece file!')

# Compute the ratio of the input over the output of the flat field
# step
ratio = wfile.data / ffile.data

# If WIM and flat field reference file provided, check that the ratio
# equals the flat field reference file. If WSM, check that the ratio
# is unity.
if wim:
```

```
    equal_check = np.isclose(ratio, flat_ref.data, atol=1e-15)
else:
    equal_check = np.isclose(wfile.data, ffile.data, atol=1e-15)

# Find bad pixel locations if any for follow up, and get the science
# and DQ array values. Check if the science array is 0 and the DQ
# array contains bits 1 and 2. If so, these are okay. Make a note
# of any other pixels.
bad = np.where(equal_check == 0)
if len(bad[0]) > 0:
    sci_bad = wfile.data[bad]
    dq_bad = wfile.dq[bad]

    for j, _ in enumerate(sci_bad):
        if sci_bad[j] == 0:
            bits = get_bits(dq_bad[j])
            if all(x in bits for x in [1, 2]):
                print(f'\tPixel (y, x) = ({bad[0][j]}, {bad[1][j]})
 was saturated in all reads.')
            else:
                prRed(f'\tPixel (y, x) = ({bad[0][j]}, {bad[1][j]})
 should be checked!')
        else:
            prRed(f'\tPixel  (y,  x)  =  ({bad[0][j]},  {bad[1][j]})
 should be checked!')
else:
    print('Test passed!')
```