rad Documentation

Release 0.19.4.dev4+g8db8b25

STScI <help@stsci.edu>

May 08, 2024

CONTENTS

I Included Resources 3

II Developer Resources 15

III Index 35

The Roman Attribute Dictionary (RAD) is a package which defines schemas for the Nancy Grace Roman Space Telescope files. Metadata attributes are used by both Science Data Processing and the Archive Keyword Dictionary.

Note: These are schemas for the ASDF file file format, used by ASDF to serialize and deserialize data for the Nancy Grace Roman Space Telescope.

Part I

Included Resources

The following are listings of all the schemas provided by this package for ASDF.

Note: Typically, schemas are used in ASDF via their tag, which can be found in the manifest.

CHAPTER

ONE

SCIENCE PRODUCTS SCHEMAS

1.1 Level 1 (uncalibrated) File schema

1.2 Level 2 (calibrated exposure) File schema

1.3 Level 3 (resampled mosaic) File schema

1.4 Level 4 File schemas

1.5 Tags

CHAPTER

TWO

REFERENCE FILE SCHEMAS

Nancy Grace Roman Space Telescope Reference Files:

CHAPTER

THREE

MANIFESTS

The `tag` manifests:

CHAPTER

FOUR

GROUND TESTS FILE SCHEMAS

4.1 FPS

4.2 TVAC1

4.3 Tags

fps/basic-1.0.0 # fps/cal_step-1.0.0 # fps/common-1.0.0 # fps/exposure-1.0.0 # fps/exposure_type-1.0.0 # fps/groundtest-1.0.0 # fps/guidestar-1.0.0 # fps/ref_file-1.0.0 # tvac/basic-1.0.0 # tvac/cal_step-1.0.0 # tvac/common-1.0.0 # tvac/exposure-1.0.0 # tvac/exposure_type-1.0.0 # tvac/groundtest-1.0.0 # tvac/guidestar-1.0.0 # tvac/ref_file-1.0.0

Part II

Developer Resources

CHAPTER

FIVE

CREATING A NEW SCHEMA

This is intended to be a quick guide to how to create a new schema in RAD. It is not intended to be a comprehensive guide, but rather a quick guide that highlights all the important considerations for RAD schema creation.

5.1 Before you begin

Before you start writing your schema, you should have a clear idea of what data you wish to store in a Roman file and how you want to organize it. Remember that RAD supports a hierarchical data model, so you can topically organize your data under headings and subheadings (or even deeper) as you see fit. In particular, you should have a clear idea of the following:

1. The name of the schema you wish to use. Be sure to determine the version number of the schema. Typically, if the schema is new, it will be `1.0.0`. So the schema name will be `<name>-1.0.0.yaml`.
2. Where in RAD you wish to locate your schema. For example if it is for a reference file then it should be located in the `reference_files` directory.
3. The keywords for your fields and their hierarchical organization.
4. The data types of all the fields that you wish to store. In particular, you need to pay attention to the following:
    - Which fields will be primitive data types like `int`, `float`, `str`, or `bool`. In JSON-schema these will be `integer`, `number`, `string`, and `boolean` respectively.
    - Which fields will require using an ASDF tag to reference another schema corresponding to a non-primitive type. In particular, you need to know if that tag is a RAD tag or an external ASDF tag.
5. Which fields will be required and which will be optional.
6. What order you want your fields to appear in the ASDF file.

Note: An ASDF tag will be defined within the tag manifest for the schema package, which adds ASDF support for the type you wish to use. For example, manifests/datamodels-1.0 is the tag manifest for the RAD package. The tag itself will be the value under `tag_uri` and the schema it references will be under `schema_uri`.

Note: All external tags should end with a `-<major version>.*` version specifier. Rather than a specific version number, this is a wildcard that will match any version of that tag. This is to ensure that the schema is not tied to a specific version of the external tag.

5.2 Create the Schema Boilerplate

After you have created your new schema file in the location under the name you have chosen, you should add the following to your blank file

```
YAML 1.1
---
$schema: asdf://stsci.edu/datamodels/roman/schemas/rad_schema-1.0.0
id: asdf://stsci.edu/datamodels/roman/schemas/<file name of schema>  # No .yaml

title: <Title of the schema>
description: |
    <A long description of the schema>

...
```

The `YAML 1.1` needs to be on the very first line of the file, while the `...` needs to be on the second to last line of the file with the final line being completely empty.

5.3 Add Your Fields

Now we will populate your schema with the fields you wish to use. In almost all cases you will want to use an `object` type for your top level of the schema, for other cases see *Alternate Ways of Adding Fields*. In this case you add the following after your `description` in the boilerplate:

```
type: object
properties:
    <first keyword>:
        title: <Title of the field>
        description: |
            <A long description of the field, can be multiline>
```

You will repeat this step for each of the top-level fields you wish to add.

5.3.1 Populate a Field's Sub-Schema

After the field's `description` at the same indentation level as the `description` keyword, you will start to add the sub-schema for the field. There are several different possibilities at this point:

- **Primitive type.**

  Things like `int`, `float`, `str`, or `bool`. In this case you will add the following:

  ```
  type: <type>
  ```

Note: The `<type>` for a Python `float` is `number` and the `<type>` for a Python `bool` is `boolean`. While the `<type>` for a Python `int` is `integer` and the `<type>` for a Python `str` is `string`.

- **Tagged type.**

  Things that are referenced via an ASDF tag. In this case you add the following:

```
tag: <tag_uri>
```

If you want to narrow the tag further than its general schema you add after the tag (at the same indentation level):

```
properties:
  <narrowed key from tag>: <schema information to narrow the key>
```

Note: If you say want to narrow an `ndarray` to a specific datatype and number of dimensions you would add the following:

```
properties:
  datatype: <dtype of the ndarray>
  exact_datatype: true
  ndim: <number of dimensions of the ndarray>
```

RAD requires that both `datatype` and `exact_datatype: true` be defined for `ndarray` tags. The `exact_datatype: true` prevents ASDF from attempting to cast the datatype to the one in the schema, meaning that if the dtype is not a perfect match to the schema a validation error will be raised.

•Dictionary-like type.

These are things that nest further fields within them. In this case you add:

```
type: object
properties:
  <first keyword>:
    title: <Title of the field>
      description: |
        <A long description of the field>
```

And then repeat the process of adding the sub-schema for each of the fields.

•List-like type.

These are lists of the same type of item. These are called an `array` in the schema, meaning that you add the following:

```
type: array
items:
  type: <type>
```

If further narrowing is required you can narrow them just like you would a tag. If you create an object or another array you likewise add the metadata in the same way as if it were a top-level field only indented appropriately.

5.3.2 Special Field Considerations

There are a few special considerations that you might need to take into account when creating your schema:

- **Enum.**

  If you have a field that can only take on a specific set of values, you can use the `enum` keyword to specify the possible values. For example:

```
enum: [<value1>, <value2>, <value3>]
```

- **Multiple Possibilities.**

  If a field can take on multiple different types, you can use the `oneOf` combiner to specify the different possibilities. For example:

```
oneOf:
  - type: <type1>
  - type: <type2>
  - type: <type3>
```

  where further metadata can be added to each of the types as needed.

  Note: Sometimes you might want to have a field which is required, but which may not take on any values at all. In this case you can use the `null` type as one of the possibilities in the `oneOf` combiner.

5.4 Add Required and Ordering Information

After you have added all of your fields, you will want to add the required and ordering information. This is done at the same indentation level as the `properties` keyword, at the end of the right before the `...`. This looks like the following:

```
required: [<required field 1>, <required field 2>, <required field 3>]
order: [<field 1>, <field 2>, <field 3>]
```

5.5 Tag Your Schema

In most cases, you will want to tag your schema with the RAD tag manifest. This performs several useful tasks:

1. It makes the object your schema represents independently (from any other RAD objects) serializable and de-serializable to ASDF.
2. It flags the object within the human-readable header of the ASDF file using the tag. This is useful for quickly identifying the type of object and differentiating otherwise identical objects.
3. It allows ASDF to easily search back into the schema from a data file to read out metadata about the object contained within the schema.
4. Allows for the use of "tag", `tag:` references as opposed to JSON-schema references. This type of reference adds additional data validation.

To tag your schema, you will need to add an entry to the RAD tag manifest, manifests/datamodels-1.0. To do this you will need to add the following after the `tags:` keyword in the manifest file (before the end `...`):

```
- tag_uri: <tag_uri>
  schema_uri: <schema_uri>
  title: <Title of the schema>
  description: |-
    <A long description of the schema>
```

Where <tag_uri> is the tag you wish to use and <schema_uri> matches the id in your schema file. If a schema is tagged, it should have

```
flowStyle: block
```

Added on the line before the ... in the schema file. This is to ensure that ASDF will write the human-readable in the file in a human-readable format.

Warning: While not explicitly necessary, RAD recommends that your formulate your file name, schema_uri, and tag_uri following standard convention. This is to avoid confusion and to make it easier to find the schema and tag and determine the associations between them. The convention is to use:

1. Ignoring the file handle (which should always be .yaml), the file name should be the path to the schema file with root being the rad/resources directory. E.g. schemas/reference_files/dark-1.0.0 or schemas/aperture-1.0.0.
2. The "version" of the schema should be the suffix of the file name having the form -<major>.<minor>.<patch>. E.g. -1.0.0.
3. The schema_uri should be the same as the file name file name described above with the RAD URI prefix asdf://stsci.edu/datamodels/roman/. E.g. asdf://stsci.edu/datamodels/roman/schemas/reference_files/dark-1.0.0 or asdf://stsci.edu/datamodels/roman/schemas/aperture-1.0.0.
4. The tag_uri should match the schema_uri with the schemas replaced with tags. E.g. asdf://stsci.edu/datamodels/roman/tags/reference_files/dark-1.0.0 or asdf://stsci.edu/datamodels/roman/tags/aperture-1.0.0.

Note: There are some cases where you might not want to tag a schema. These are generally, when the schema is not intended to be used as a standalone object. This can be the case when the schema is intended to be extended by another schema, see *Pseudo Inheritance* for more information.

5.6 Alternate Ways of Adding Fields

There are two additional ways that one might formulate the top level of a schema which do not involve using an object type (*Pseudo Inheritance* is also a method but it still involves objects). These are when one needs to tag a specially defined list (array) data or when one needs tag a scalar type. In both these cases, the schema is acting to mix metadata into the schema in a way that can be reused in other schemas rather than to define a standalone object.

Aside from reuse this is done so that ASDF can correctly search and pull metadata from the underpinning schemas. This is largely due to the difficulty in having ASDF traverse through multiple layers of allOf combiners in its search and find efforts in the schemas. These combiners are largely the results of *Pseudo Inheritance*. By having a tag ASDf is able to bypass the recursive search and jump directly to the schema that is being referenced.

5.6.1 Tagged List

Currently, there is only one case where the RAD schemas tag a list, the schemas/cal_logs-1.0.0 schema. Just as in this case, the top level of the schema will be:

```
type: array
items:
  - <sub-schema(s) describing items>
```

The `items` simply contains a bulleted list of the sub-schemas that describe possibilities for the items in the list.

5.6.2 Tagged Scalar

The other case is when one needs to tag a scalar type. This is mostly to help with the ASDF metadata searching. All such schemas need to be inside the `schemas/tagged_scalars` directory so that the correct Python data nodes can be automatically constructed for the data models.

In this case, you add the following after the schema description if the type of the scalar is a primitive type:

```
type: <primitive type>
```

However, if the scalar is planned to be represented by a non-primitive type such as a time or some other special type, then you will need to use a `$ref` back to the `schema_uri` not `tag_uri` for the schema that describes this type. It is important to use the `schema_uri` because referencing a `tag_uri` will cause ASDF validation to not only check that the data is valid for the schema, but also that the type being used is exactly one of the types associated with that tag (sub-classes will fail validation in this case). Since the ASDF extension supporting that type is outside of RAD's control, it is not possible for it to even know about RAD's sub-classes and so this will not work. Hence, a `$ref` to the `schema_uri` is necessary. This needs to be added after the description of the schema using:

```
allOf:
  - $ref: <schema_uri>
```

The `allOf` combiner is necessary because of quirks in how JSON-schema actually functions; meaning that for ASDF 3.0+ to correctly handle the schema without issues, the `allOf` combiner is necessary, see PR 222 for more details.

5.7 Testing Schemas

Once you created a schema, run the tests in the `rad` package before proceeding to write the model.

Note: The schemas need to be committed to the working repository and the `rad` package needs to be installed before running the tests.

5.8 Creating a Data Model

The DataModel objects from RDM which act as the primary outward facing Python interface to the data described by the RAD schemas are simply wrappers around the actual data container objects. As such these DataModel objects are not directly defined by anything in RAD. However, they are closely related to the RAD schemas. As such, certain additional things are added to some schemas to make this relationship between DataModel objects and some schemas more clear.

First, note that since all the schemas in RAD are hierarchical, there eventually will exist a "top-level" schema which acts to describe all the data that is expected to be in a given ASDF file for Roman. Since each ASDF file will correspond to a specific DataModel object and those objects are wrappers around the actual data container objects, that "top-level" schema effectively describes the data structure of a given DataModel object. Hence, this "top-level" schema should be called out in a way that makes it clear that it is the schema which fully describes the structure of a DataModel and its associated Roman ASDF file.

To do this, right after the description of the schema in the schema file, the following should be added:

```
datamodel_name: <name of the datamodel in Python>
archive_meta: None
```

The datamodel_name field is simply so that we can test that a DataModel exists for each "top-level" schema and that each of these schemas maps to exactly one DataModel. Moreover, it documents which DataModel maps to which schema as this is not always completely clear due to the fact that the schema names and DataModel names do not follow a strict naming pattern.

The archive_meta field is a placeholder for future use. It is intended to allow the archive to add additional metadata about specific Roman ASDF files, which do not fit neatly into the metadata structures it uses for describing the fields of in the schemas, see *External Metadata* for more details.

5.9 Pseudo Inheritance

When creating schemas, there are cases in which you might want multiple schemas to share identical structures, but do not want to repeat this information in multiple places. Since JSON-schema does not support inheritance in the "classical" sense, we have to employ a workaround. This workaround employs the JSON-schema allOf combiner together with the JSON-schema reference keyword, $ref. This results in a schema code block that looks like the following:

```
allOf:
  - $ref: <schema_uri>
  - type: object
    properties:
       <additional properties to add to existing schema>
```

This acts somewhat like inheritance because it requires that the data described by the schema must satisfy the requirements of the schema being referenced and the additional new object included in the allOf combiner.

This method of combining schemas maybe used at the top level of a schema in order to create a full inheritance-like relationship or it may be used in some sub-schema to do a similar thing. In any case, this should be the only usage of the $ref keyword in the schema file.

5.10 External Metadata

In addition to describing the data structure of Roman ASDF files, RAD also acts to house metadata about how the Roman ASDF files are to be interacted with. This "external metadata" is not directly related to the structure of the data structure itself, but rather describes how the data contained within that structure will be integrated into the archives or how some of that data was created external to the Romancal pipeline.

Currently, there are two types of external metadata that are supported by RAD:

1. `sdf`
2. `archive_catalog`

5.10.1 sdf

This is the metadata given to fields which are populated by the SDF software before the data is processed by the Romancal pipeline. This metadata currently consists of two fields:

1. `special_processing`: which is a string that describes the special processing that was done to create the data in SDF.
2. `source`: which is a string that describes the source of the data used by SDF.

Both of these values are typically provided to us by the SDF software teams and thus should be done in consultation with them. If the SDF software teams have not indicated the values yet then the fields should be filled with `VALUE_REQUIRED` and `origin: TBA` respectively.

5.10.2 archive_catalog

This is the metadata given to fields that will be incorporated into the archive to describe the Roman ASDF file. This metadata consists of two fields:

1. `datatype`: which describes the datatype of that will be used by the archive's database to store the data contained within the field. This maybe things such as if its a string and if so how long or what type of number it will be.
2. `destination`: This is a list of strings of the form `<table name>.<column name>`, which describe where that data will be stored in the archive's database. Typically `<column name>`, will match the keyword of the field in the schema. This is not always the case as sometimes multiple fields from different parts of the files may end up in the same table, but whose keywords are the same. When this occurs, the archive will inform us of what the correct `<column name>` should be. The `<table name>` is the name of the table in the archive's database and is typically provided to us by the archive to be recorded in the schema.

In both cases, the metadata should be added in consultation with the archive team. This includes if the field should even be included into the archive.

CHAPTER

SIX

CONTRIBUTING

We welcome feedback and contributions of all kinds. Contributions of code, documentation, or general feedback are all appreciated.

6.1 Reporting Issues

Feature requests and bug reports can be posted at Roman Attribute Dictionary's github page.

6.2 Contributing Code and Documentation

We love contributions! If you're interested in contributing to this project, please open a Pull Request or issue and we would be glad to work with you.

CHAPTER

SEVEN

CHANGE LOG

7.1 0.20.0 (unreleased)

-

7.2 0.19.4 (2024-05-08)

- Updated RTD with documentation for new data products. [#419]

7.3 0.19.3 (2024-04-25)

- Duplicated the keywords from groundtest to tvac_groundtest. [#409]

7.4 0.19.2 (2024-04-17)

- Duplicated the keywords from base_exposure to exposure and similarly for base_guidestar and guidestar. [#406]

7.5 0.19.1 (2024-04-04)

- Add new schemas to documentation. [#386]
- Convert tag keywords to wildcards for external tags. [#370]
- Added `exact_datatype` arguments to prevent ASDF from casting array datatypes during save. [#369]
- Add documentation on how to create a new schema. [#375]
- Add `FPS` and `TVAC` schemas. [#364]
- Update titles and descriptions to those provided by INS. [#361]
- Updated product table names. [#382]
- Changed image units from e/s to DN/s (and added support for MJy/sr). [#389]
- Add attributes under the `basic` schema to `WfiMosaic.meta`. [#390]
- Split cal_step into L2 and L3 versions. [#397]
- Add Members Keyword to Resample Schema. [#396]

- Create the flux step schema. [#395]
- Create `outlier_detection` schema and add bit mask field to both it and `resample`. [#401]
- Add source_catalog and segmentation_map schemas for Level 2 and Level 3 files. [#393]

7.6 0.19.0 (2024-02-09)

- Added streamlined Level 3 Mosaic metadata schemas. [#334]
- Remove the unused `variance-1.0.0` schema. [#344]
- Add wcs tag to wfi_image and wfi_mosaic schemas. [#351]

7.7 0.18.0 (2023-11-06)

- Added Slope and Error to Dark reference schema. [#323]
- Removed `err` array from dark schema. [#324]
- Expanded origin db string length. [#326]
- Updated minimum python version to 3.9. [#325]
- Added truncated keyword. [#330]
- Added GuideWindow db table to Basic tagged scalars. [#327]
- Added optional dq array. [#328]
- Update required elements for release. [#337]

7.8 0.17.1 (2023-08-03)

- Added "archive_catalog" field to ref_file. [#303]
- Added a prefix `s_` to the archive destination in "cal_step". [#303]
- Require all the new `cal_step` steps to be present in the `cal_step` schema. [#301]
- Add missing unit enforcements to various schemas. [#300]

7.9 0.17.0 (2023-07-27)

- Fix invalid uri fragment in rad_schema. [#286]
- Update the steps listed in `cal_step` to reflect the currently implemented steps. The new additions are `outlier_detection`, `refpix`, `sky_match`, and `tweak_reg`. [#282]
- Update the steps listed in `cal_step` with the `resample` step. [#295]
- Fix the URIs for `inverselinearity` and add consistency checks for names/uris. [#296]
- Add `archive_meta` keyword for the MAST archive to encode information specific to the archive's needs. [#279]

7.10 0.16.0 (2023-06-26)

- Fix minor discrepancies found when looking over the schemas. [#267]
- Bugfix for `inverse_linearity-1.0.0`'s `reftype` so that it is CRDS compatible. [#272]
- Add schema `refpix-1.0.0` as a schema for the reference pixel correction's reference file. [#270]
- Add keyword to indicate if and which datamodel the schema describes. [#278]
- Add schema `msos_stack-1.0.0` as a level 3 schema for SSC. [#276]

7.11 0.15.0 (2023-05-12)

- Update program to be a string to match association code [#255]
- Add gw_science_file_source to GW file, update size of the filename [#258]
- Update program to be a string to match association code [#255]
- Update guide star id, add catalog version, and add science file name [#258]
- Add gw_science_file_source to GW file, update size of the filename [#258]
- Remove use of deprecated `pytest-openfiles` `pytest` plugin. This has been replaced by catching `ResourceWarning` s. [#231]
- Add read pattern to the exposure group. [#233]
- Add `distortion` keyword option to the list of reference files, so that the `distortion` reference file can be properly allowed in by the `ref_file-1.0.0` schema. [#237]
- Changelog CI workflow has been added. [#240]
- Clarifying database tables for guidewindows and guidestar." [#250]
- Remove the `unit-1.0.0` schema, because it is no-longer needed. [#248]
- Remove the unused `pixelarea-1.0.0` schema, which was replaced by the `reference_files/pixelarea-1.0.0` schema. [#245]
- Added support for level 3 mosaic model. [#241]
- Add further restrictions to the `patternProperties` keywords in the `wfi_img_photom` schema. [#254]

7.12 0.14.2 (2023-03-31)

- Format the code with `isort` and `black`. [#200]
- Switch linting from `flake8` to `ruff`. [#201]
- Start using `codespell` to check and correct spelling mistakes. [#202]
- Created inverse non-linearity schema. [#213]
- Added PR Template. [#221]
- Begin process of decommissioning the Roman specific, non-VOunits. [#220]
- Fix schemas with $ref at root level. [#222]
- Add schema for source detection. [#215]

- Temporarily make source detection optional in cal_logs. [#224]
- Add database team to Code Owners file [#227]
- Update CodeOwners file [#230]

7.13 0.14.1 (2023-01-31)

- Update guidwindow titles and descriptions. [#193]
- Changed science arrays to quantities. [#192]
- Add units to the schemas for science data quantities to specify allowed values. [#195]
- Update Reference file schemas to utilize quantities for all relevant arrays. [#198]
- Fix `enum` bug in schemas. [#194]
- move metadata to `pyproject.toml` in accordance with PEP621 [#196]
- Add `pre-commit` support. [#199]
- Add IPC reference schema. [#203]
- Updated the variable type of x/y start/stop/size in guidewindow and guidestar schemas. [#205]
- Changed SDF "origin" in ephemeris-1.0.0.yaml to use definitive/predicted ephemeris. [#207]
- Adjust activity identifier in observation schema to better reflect potential values. [#204]
- Deleted source_type_apt from target-1.0.0.yaml [#206]
- Add reftype to IPC Schema. [#214]

7.14 0.14.0 (2022-11-04)

- Use PSS views in SDF origin attribute. [#167]
- Add support for specific non-VOUnit units used by Roman. [#168]

7.15 0.13.2 (2022-08-23)

- Add `IPAC/SSC` to `origin` enum. [#160]
- Add archive information to `ref_file` and fix indentation there. [#161]

7.16 0.13.1 (2022-07-29)

- Removed CRDS version information from basic schema. [#146]
- Changed the dimensionality of the err variable in ramp. [149#]
- Create docs for RTD. [#151]
- Moved gw_function_start_time, gw_function_end_time, and gw_acq_exec_stat from GuideStar to GuideWindow. Removed duplicate gw time entries. [#154]

- Changed optical filter name W146 to F146. [#156]
- Moved archive related information in the `basic` schema directly into a tagged object for easier retrieval by ASDF. [#153, #158, #159]
- Fix ref_file schema. [#157]

7.17 0.13.0 (2022-04-25)

- Remove start_time and end_time from the observation schema [#142]

7.18 0.12.0 (2022-04-15)

- exposure schema update in include descriptions [#139]
- Moved ma_table_name and ma_table_number from observation to exposure schemas. [#138]

7.19 0.11.0 (2022-04-06)

- Initial Guide Window Schema [#120]
- Enumerate aperture_name in the aperture schema [#129]
- Remove exptype and p_keywords from Distortion Model [#127]
- Added photom keyword attribute to cal_step schema. [#132]
- Added ma_table_number to observation and dark schemas. [#134]
- Create distortion schema [#122]

7.20 0.10.0 (2022-02-22)

- Moved detector list to new file for importing to both data and reference schemas. [#119]
- Added support for Distortion reference files. Tweaked schema for WFI detector list. [#122]
- Changed input_unit and output_unit keyword types, titles, and tests. [#126]
- Removed exptype and p_keywords from Distortion schema. [#128]

7.21 0.9.0 (2022-02-15)

- Add FGS (Fine Guidance System) modes to guidestar schema. [#103]
- Set all calsteps to required. [#102]
- Added p_exptype to exposure group for reference files (dark & readnoise) to enable automatic rmap generation. Added test to ensure that the p_exptype expression matched the exposure/type enum list. [#105]
- Added boolean level0_compressed attribute keyword to exposure group to indicate if the level 0 data was compressed. [#104]

- Update schemas for ramp, level 1, and 2 files to contain accurate representation of reference pixels. The level 1 file has an array that contains both the science and the border reference pixels, and another array containing the amp33 reference pixels. Ramp models also have an array that contains the science data and the border reference pixels and another array for the amp33 reference pixels, and they also contain four separate arrays that contain the original border reference pixels copied during the dq_init step (and four additional arrays for their DQ). The level 2 file data array only contains the science pixels (the border pixels are trimmed during ramp fit), and contains separate arrays for the original border pixels and their dq arrays, and the amp33 reference pixels. [#112]
- Added `uncertainty` attributes to `photometry` and `pixelareasr` to the photometry reference file schema. [#114]
- Removed `Photometry` from required properties in `common`. [#115]
- Updated dark schema to include group keywords from exposure. [#117]

7.22 0.8.0 (2021-11-22)

- Add `cal_logs` to wfi_image-1.0.0 to retain log messages from romancal. [#96]

7.23 0.7.1 (2021-10-26)

- Reverted exposure time types from string back to astropy Time. [#94]

7.24 0.7.0 (2021-10-11)

- Added nonlinearity support. [#79]
- Added saturation reference file support. [#78]
- Added support for super-bias reference files. [#81]
- Added pixel area reference file support. [#80]
- Removed `pixelarea` and `var_flat` from the list of required attributes in `wfi_image`. [#83]
- Changed certain exposure time types to string. Added units to guidestar variables, where appropriate. Removed references to RGS in guidestar. Added examples of observation numbers. [#91]
- Added mode keyword to dark and readnoise. [#90]
- `RampFitOutput.pedestal` needs to be 2-dimensional. [#86]
- Added optical_element to appropriate reference file schemas. Added ma_table_name to dark schema. Adjusted pixelarea schema imports. [#92]

7.25 0.6.1 (2021-08-26)

- Changed ENGINEERING to F213 in optical_element. [#70]
- Workaround for setuptools_scm issues with recent versions of pip. [#71]

7.26 0.6.0 (2021-08-23)

- Added enumeration for `meta.pedigree`. [#65, #67]
- Added more steps to the cal_step schema. [#66]

7.27 0.5.0 (2021-08-06)

- Adjust dimensionality of wfi_science_raw data array. [#64]
- Added dq_init step to cal_step. [#63]

7.28 0.4.0 (2021-07-23)

- Removed basic from ref_common and moved some of its attributes directly to ref_common [#59]
- Updated dq arrays to be of type uint32. Removed zeroframe, refout, and dq_def arrays. [#61]

7.29 0.3.0 (2021-06-28)

- Updated rampfitoutput model and WFIimgphotom models. Renamed rampfitoutput ramp_fit_output. [#58]

7.30 0.2.0 (2021-06-04)

- Updated yaml files to match latest in RomanCAL. [JIRA RCAL-143]
- Changed string date/time to astropy time objects. [JIRA RCAL-153]
- Updated id URIs. [JIRA RCAL-153]
- Updated all integers to proper integer types. [JIRA RCAL-153]
- Updated exposure.type. [JIRA RCAL-153]
- Change gs to gw in guidestar to reflect that they are all windows. [JIRA RCAL-153]
- Corrected Manifest URI. [#5]
- Removed keyword_pixelarea from Manifest. [#11]
- Removed .DS_Store files. [#7]
- Change URI prefix to asdf://, add tests and CI infrastructure. [#14]
- Moved common.yaml keywords to basic.yaml, and adjusted tests for basic.yaml. [JIRA RAD-7]

- Added misc. required db keyword attributes. [JIRA RAD-7]
- Added wfi photom schema and tests. [#34]
- Added Dark schema and updated Flat schema. [#35]
- Added dq schema. [#32]
- Added readnoise, mask, and gain schemas. [#37]
- Added support for ramp fitting schemas. [#43]
- Updated aperture, basic, ephemeris, exposure, guidestar, observation, pixelarea, and visit schemas. [#46]
- Added support for variance object schemas. [#38]

7.31 0.1.0 (unreleased)

- Initial Schemas for Roman Calibration Pipeline and SDP file generation

Part III

Index

- genindex
- modindex
- search