Challenges serializing Rubin Observatory data products

James F. Bosch[a], Tim Jenness[b], and Gregory P. Dubois-Felsmann[c]

[a]Department of Astrophysical Sciences, Princeton University, Princeton, NJ 08544, USA
[b]NSF-DOE Vera C. Rubin Observatory / NSF NOIRLab, 950 N. Cherry Ave., Tucson, AZ 85719, USA
[c]Caltech/IPAC, California Institute of Technology, MS 100-22, Pasadena, CA 91125-2200, USA

ABSTRACT

At the NSF-DOE Vera C. Rubin Observatory the image-based data products include data models that are not described by any existing FITS standard. These include spatially-varying point-spread functions, complex World Coordinate System models, provenance information, and shutter motion profiles. For Data Preview 1 these data models were written using generic C++ storage APIs, resulting in files that can only be read easily by using our software. For Data Preview 2 we have changed our approach so as to write FITS files that are more accessible by the community and can be accessed more efficiently from object stores. In this presentation we will describe our new approach to serializing data models and how it compares with the existing files, and our approach for improving object store access to individual extensions and integration with the Butler. We will also discuss how we can expand our file format support to include alternatives such as HDF5, Zarr, and ASDF.

1. INTRODUCTION

The NSF-DOE Vera C. Rubin Observatory is designed to take data for the Legacy Survey of Space and Time over a period of 10 years.[1] We made our Data Preview 1 (DP1) release in 2025 using data from our commissioning camera (LSSTComCam)[2] and we are currently preparing for Data Preview 2 (DP2) using data from the LSST Science Camera (LSSTCam).[3,4] For all the Hyper Suprime-Cam Subaru (HSC) Strategic Program (SSP) data releases[5] and for DP1 our image data products were written in FITS format following the FITS standard[6] but the FITS standard is not sufficient to fully define all the information that we need to record in our FITS files. The resulting FITS files were able to represent concepts such as variances and masks but provenance and point-spread function definitions were represented as opaque binary-table data that required Rubin-specific software[7] to interpret. In this paper we will describe the FITS layout we used for DP1 and describe its limitations. Then we will present our new approach to modeling our data structures and how we represent it in both FITS and HDF5 files.

2. THE RUBIN VISIT IMAGE DATA MODEL

One of the most complex image dataset types we write is that associated with our single-epoch visit images. At Rubin we define "visit" as an on-sky exposure. For a visit observation the resulting files have to describe the detector properties, the point-spread function, the non-linear large-scale WCS, aperture corrections and other calibration effects.

In the Rubin LSST Science Pipelines,[7] visit images for individual detectors in the LSSTCam focal plane are natively represented in a Python data model, with a set of components described below (monospace-font terms that follow are typically Python class names). Our main package implementing the legacy serialization code is called AFW* with the bulk of the implementation being in C++. This in-memory representation is serialized to an external representation using the Rubin Butler system.[8] While the Butler was designed to permit multiple serialization forms for the in-memory datasets, for the Rubin pipeline implementations used through DP1, the only supported external representation was FITS. Every visit includes the image pixel data array, along with corresponding 32-bit mask and variance arrays, together with FITS-style (keyword, value, and comment) metadata.

The following additional components then go beyond the standard elements of the FITS file data model.

*Originally this stood for "application framework" and over time people have said "astronomy framework", but we just refer to it these days as "afw."

2.1 Bounded Fields

Some of the data model components rely on being able to represent a mathematical function to be evaluated over specific regions of the image. We call these *bounded fields* and they are usually represented as Chebyshev polynomials, but they can also be specified as splines or lazy arithmetic combinations (e.g., sums or products) of multiple other bounded fields.

2.2 Visit Information

A `VisitInfo` object contains standardized information about the entire visit. This includes the weather conditions, the instrument and observatory location, the time of the observation, the boresight tracking coordinates, and the exposure time. Calibration observations also result in a `VisitInfo` being defined although some of the components may be undefined in that case, such as for observations of the internal dome screen. We will not consider calibration datasets any further in this discussion since they generally are representable by a subset of the visit data model.

2.3 Filter Label

This is a small look-up table defining the physical filter used for the observation and the waveband for that filter. In older data releases made for HSC SSP[9] using the LSST Science Pipelines we attempted to store a more complex filter type that recorded filter widths and centers. We decided that this led to discrepancies between the measured filter profiles and the data files when the calibrations of filter profiles were subsequently improved. It is much simpler to store the physical filter name itself and then use that as a look-up key into an external data set.

2.4 Detector Information

LSSTCam has 189 science detectors along with additional wavefront sensors and guider detectors. Each dataset contains a special detector object that describes the properties of this detector such as the serial number and manufacturer, the amplifier layout, and how the coordinates transform from focal plane position in millimeters to integer pixel coordinates.

2.5 Point-Spread Function

There are many ways to represent a point-spread function (PSF). We use a plugin system to allow different variants to be stored. We use both a heavily modified version of `PSFEx`[10,11] and `Piff`[12,13] in our production pipelines. The PSF can vary across the image and the PSF model has to be able to account for that.

2.6 Exposure Summary Statistics

During the analysis of a visit the pipeline calculates derived scalar quantities which we store in a small class named `ExposureSummaryStats`. These include summaries of global PSF properties, source counts, and effective exposure times and bounding boxes. This information can be used by downstream pipeline components to decide whether a visit should be included in a coadd or not. Unlike `VisitInfo` the summary statistics can change depending on pipeline configuration.

2.7 World Coordinate Systems

At the large scale of the LSSTCam focal plane combined with atmospheric effects the standard FITS approaches to specifying a world coordinate system, particularly its deviations from a nominal spherical projection, are not sufficiently rich or accurate. We use the Starlink AST WCS library[14] as this gives us the flexibility to define our WCS in terms of individual transformation steps that can be chained together in whatever order we deem appropriate. In that way we can include large scale polynomial distortions as well as more localized effects by using splines. The WCS is represented by an AST mapping that is abstracted from the Python user by a Rubin-specific interface.

```
No.    Name      Ver    Type      Cards   Dimensions   Format
  0  PRIMARY       1 PrimaryHDU     707   ()
  1  IMAGE         1 CompImageHDU     71   (4072, 4000)   float32
  2  MASK          1 CompImageHDU     93   (4072, 4000)   int32
  3  VARIANCE      1 CompImageHDU     71   (4072, 4000)   float32
  4  ARCHIVE_INDEX    1 BinTableHDU     41   93R x 7C   [1J, 1J, 1J, 1J, 1J, 64A, 64A]
  5  FilterLabel    1 BinTableHDU     28   1R x 3C   [2X, 32A, 32A]
  6  Detector      1 BinTableHDU    115   1R x 22C   [1QA(7), 1J, 1J, 1QA(13), 1J, 1J, ... 1QA(3), 1J]
  7  TransformMap    1 BinTableHDU     33   19R x 5C   [1QA(10), 1QA(7), 1QA(10), 1QA(7), 1J]
  8  ExposureSummaryStats    1 BinTableHDU     19   21R x 1C   [1QB(102617)]
  9  Detector      2 BinTableHDU    201   16R x 38C   [3X, 1QA(3), 1J, 1J, 1J, 1J, 1D, ..., 1QA(2)]
 10  SkyWcs        1 BinTableHDU     21   1R x 2C   [1QB(10399), 1QB(10437)]
 11  ApCorrMap     1 BinTableHDU     21   64R x 2C   [64A, 1J]
 12  ChebyshevBoundedField    1 BinTableHDU     41   34R x 6C   [1J, 1J, 1J, 1J, 1J, 1D]
 13  ChebyshevBoundedField    2 BinTableHDU     42   31R x 6C   [1J, 1J, 1J, 1J, 1J, 9D]
 14  PhotoCalib    1 BinTableHDU     36   1R x 5C   [1X, 1D, 1D, 1J, 1J]
 15  Polygon       1 BinTableHDU     21   5R x 2C   [1D, 1D]
 16  PsfexPsf      1 BinTableHDU     52   1R x 9C   [1J, 1J, 1J, 1J, 1J, 1J, 1D, 1D, 1E]
 17  PsfexPsf      2 BinTableHDU     46   1R x 8C   [2J, 1J, 6D, 6D, 3J, 41334E, 2D, 2D]
```

Figure 1. The structure of a representative DP1 FITS file using the legacy schema.

2.8 Validity Region

This is a polygon in pixel coordinates defining the valid region of the image. For most detectors, the valid polygon is just the detector's bounding box, but those on the edge of the focal plane have a valid polygon boundary that marks where vignetting occurs.

2.9 Aperture Corrections

Multiple aperture corrections can be stored, each one being labeled and associated with a bounded field. Different measurement algorithms can result in different aperture corrections and the user can select the appropriate one. Each photometry algorithm has its own aperture correction, which is used to tie that algorithm to a common photometric system.

2.10 Photometric Calibration

Part of the visit processing determines how the image should be corrected to convert detector units to nJy. The calibration can vary across the field and is represented by a bounded field.

2.11 Class Hierarchy

In the LSST Science Pipelines every image dataset is represented by the `lsst.afw.image.Exposure` class in both C++ and Python. An `Exposure` consists of a `MaskedImage` and an `ExposureInfo`. The `MaskedImage` contains two `Image` objects (including the variance) and a `Mask` object, and bounding boxes and metadata. The image WCS is associated with the `ExposureInfo` where the additional components described above are defined. This design decision has caused confusion over the years, because the consequence is that you cannot associate world coordinates with a `MaskedImage` or `Image`, even though, conceptually, everyone expects such entities to have a well-defined WCS.

3. DP1 FITS FILES

A summary of the layout of a FITS file produced for a visit image in DP1 is shown in Fig. 1 and shows the extensive use of FITS binary tables. The pixel data use standard FITS data arrays with compression to represent the primary image data, the variance, and the mask. The mask image is a 2D 32-bit integer array supporting 32 individual mask planes. A set of common mask planes is hard-coded into AFW itself to enforce standardization of the core planes, while others are defined by pipeline tasks that add them to the images they write out.

This means that the assignment of the per-pixel mask Booleans to bit positions is contingent on the specific configuration of the processing pipeline. We provide headers that associate symbolic names with bit positions in order to mitigate the impact of this; users in downstream code are expected to refer to specific mask Booleans by their symbolic names, not their pixel positions.

The FITS primary HDU's header contains the standard header values from the raw data file[15] but a visit image file also adds headers itself on serialization. We follow the common community convention of having no data payload in the primary HDU (NAXIS=0), leaving all the content to extension HDUs. As noted above, in general it is not possible to accurately represent the distortions in the world coordinates of a visit image using the standard FITS-WCS approaches. We therefore store an FITS-standard approximation of the WCS in the headers using the SIP "registered convention"[16] so that tools like Astropy[17] and IPAC Firefly[18] can display sky coordinates and perform catalog overlays on our images. For DP1 these approximations were good to a few milliarcseconds, i.e., around the astrometric precision of the true WCS, but this may not always be true in the future.

The `VisitInfo` struct also serializes into the FITS primary header. We now consider this to be an anti-pattern because the keywords are not namespaced. This can lead to confusion where a header from an instrument using `PRESSURE` to represent the air pressure in one unit will get a new value in a potentially different unit. Even more dangerously, we write the midpoint of the observation using the standard `DATE-AVG` header and we force `TIMESYS` to TAI without attempting to adjust any other standard `DATE-` headers that might be in the file from the instrument raw headers using UTC. This can lead to errors from any tooling that is trying to interpret headers using the standard. If we were approaching this today we would store the `VisitInfo` in the header using an approach like `HIERARCH LSST VISITINFO DATE-AVG` and make it absolutely clear which headers were coming from the serialization and which were originally from the raw data. We also write the detector name and detector serial ID into headers `DETNAME` and `DETSER`.

The remaining content is stored in FITS binary tables.

3.1 Table Archive System

The fundamental serialization approach for the DP1 data involves a generic persistence infrastructure that converts exposure components into rows of tables that can be stored in FITS.

Each component registers itself as a `Storable` that implements the creation of a tabular version of the object that can be stored. During serialization each component is allocated an integer archive ID that is referenced by a table of contents stored in the `ARCHIVE_INDEX` extension. The ID itself is stored in the primary FITS header with key `ARCHIVE_INDEX_<COMPONENT>`.

Each component is then represented in the `ARCHIVE_INDEX` table indexed by the ID which refers to a later HDU and rows within that HDU. If components share the same table schema they are written to the same HDU, which leads to a storage efficiency gain but does mean that each HDU can not be interpreted without the primary index given that the HDU name reflects the order that components were serialized in and so can vary from file to file.

3.1.1 Component Serializations

The components themselves are serialized in different ways.

FilterLabel This is a very simple table with two rows. One row indicates the wave band (and whether it is defined or not) and the other row contains the physical filter name (and its definedness state). These two items could have easily been represented as FITS headers, as was done for `VisitInfo`, but instead we took the more standardized yet complex approach of fitting directly into the archive system by creating a binary table.

Detector This is written as two catalogs. The first summarizes detector properties and the second describes amplifier properties with one row per amplifier.

TransformMap This contains 2-D mappings that define detector transformations from different frames. The mappings are byte string native serializations of the AST library mappings.

Polygon Polygons are represented as a two column table containing the `x` and `y` vertices.

SkyWcs This represents the pixel to sky transformations and is stored in the table as a byte string using the native string erialization provided by the AST library. Optionally we can also store the FITS WCS approximation of the full mapping.

PhotoCalib This table contains the scalar calibration information and references to archive IDs of bounded fields.

ApCorrMap This is a simple table that associates the name of the correction with a reference to the bounded field.

ChebyshevBoundedField These define the bounded fields referenced from other HDUs. Each row defines the Chebyshev order, the bounding box for which the polynomial is valid and the polynomial coefficients. In the example output there are two HDUs labeled as Chebyshevs because there are different coefficient counts in use.

ExposureSummaryStats This is a Python data class. All Python types must serialize as strings which are stored in a single bytes column. For this class the serialization uses YAML as a single string column and since this is a common serialization schema many components can share this table.

PSFs The PSF component is polymorphic. AFW only sees it through the persistable interface. The actual FITS representation depends on the concrete PSF class and its registered factory. For example, `PsfexPsf` from `meas_extensions_psfex` writes two catalogs and other PSF subclasses use their own schemas. The PIFF serialization uses the Python AFW `Storable` interface and writes a single bytes column from the output of the Python `pickle` interface.[19,20] At the time of creation of DP1 there was no standardized PIFF serialization we could use outside of writing a standalone FITS file other than `pickle` (a situation which has now been fixed). This decision was the pragmatic one at the time but it means that a user reading our files without using our code *must* use Python to reconstruct the PIFF object and does not have the option of implementing their own equivalent in another language. The use of `pickle` format also makes it impossible to document these files in a way that is useful to an archive user long in the future, as well as posing documented IT security issues (`pickle` readers are vulnerable to arbitrary-code-execution exploits).

4. LIMITATIONS OF THE DP1 SERIALIZATION

We have been battling the limitations of this serialization framework for some time.[21,22] A key limitation for implementers was the difficulty in adding new components to the archive system. The core of the system requires C++ and it was only relatively recently that we were able to include Python classes into the serialization mechanism. Even then, each class is required to inherit from the `Storable` infrastructure in order to hook into the serialization system. This adds an unnecessary burden on each class. Furthermore, since we expect every image dataset we write to be using the AFW `Exposure` classes this means that whenever we need to add support for a new component, every dataset type sees that component even if it is not useful for that variant.

Whilst the implementation complexity is only of concern to developers and is a one-off cost, a more fundamental issue relates to the long-term usability of these files. As can be seen from Fig. 1 the extension names do not necessarily tell you what the extension is associated with; for example, bounded fields are serialized as extensions but you do not know which component the bounded field is associated with. As described previously, multiple components can also be written to the same HDU, with the `EXTNAME` referring to only one of the components out of convenience. In order to understand what each extension is part of you must first read the index extension.

The complex logic effectively requires an end-user to use the `lsst.afw` Python classes to read the extensions. Re-implementing the logic in Astropy or a language other than Python or C++ is potentially possible for a highly motivated individual (and with modern large-language model coding tools using the AFW code as a reference

someone might be tempted to try it), but it would be preferable to avoid this situation (and the associated need to create corresponding non-AFW classes for all the components) if at all possible. Additionally, we are formally required to provide complete documentation on how to read the files we distribute to the community. This motivates us to choose a file format that is easy to describe and understand. Using `pickle` for the current format makes documentation effectively impossible.

The requirement to read the archive index does have other ramifications. The system was designed in an era when there was an assumption that the file would always be available on a POSIX file system. For DP1 the data are stored in Google Cloud Storage. DP2 will be stored at the SLAC National Accelerator Laboratory but accessed remotely from Google[23] using web protocols, with some data cached at Google. When a user requests a component, such as the valid polygon, we would ideally like to retrieve only the HDU containing that information. The simplest option would be to download the entire file to local storage and then read the component. To avoid downloading megabytes of pixel data, instead we construct fake single-pixel image HDUs for the image/variance/mask arrays and then download all the extension HDUs so we can write a file locally and use the AFW code to read the index and then construct the extension. This is not efficient but is an inevitable outcome of the approach.

5. CHANGING APPROACH FOR DP2

Given the discussion above, we decided that for DP2 we would change the way we represent image datasets in Python and also change how we serialize them. Our main requirements were:

- The main implementation language should be Python rather than C++.
- Each image dataset type (such as visit images, difference images, coadds) should be represented by its own Python class. This allows each dataset type to define only the components that are directly relevant to that type. It also allows more type safety in the pipelines rather than always assuming a generic image container.
- Informed by the experience with the Advanced Scientific Data Format (ASDF),[24] the data models written to the file must be associated with public formal schemas.
- It should be possible to locate and read a component efficiently.
- People should be able to install software to read the files with our data models with only a `pip install` and the FITS files must be usable from Astropy and general viewing tools.
- It should be possible to document the file format in natural language sufficiently clearly that the data files, if preserved, will remain usable far into the future even in the absence of the ability to run code developed in the 2020s. We are conscious of the very long-term value to time-domain astronomy of the Rubin data even in future eras in which their depth or accuracy may have long since been superseded.

One additional input into the process was to make a conscious effort to distinguish the data model from the serialization format. Separating the model from the file format has been a long-running debate in the community[25,26] as we discuss the limitations of the FITS format[27] and the possibilities for being more flexible in terms of archival artifacts being produced.

5.1 The New Data Model

The most fundamental part of the data model is the `Image` class. This contains a two-dimensional NumPy data array with a pixel origin and units, a projection that can map pixels to world coordinates, as well as some simple keyword/value metadata support.

There is also a `Mask` class which is similar to `Image` except it has an additional schema that describes the mask planes (bit numbers, mask name, mask description) and internally is represented as a three-dimensional array even though the external interface is a two-dimensional array. The third dimension uses unsigned bytes

Listing 1. PIFF solution in Pydantic model referencing a data component.

```
"interp": {
  "tables": {
    "solution": {
      "metadata": {},
      "table": {
        "columns": [
          {
            "data": {
              "source": "ndf:/MORE/LSST/PSF/PIFF/INTERP/SOLUTION/DATA_ARRAY/DATA",
              "shape": [
                625,
                15
              ],
              "datatype": "float64",
              "byteorder": "big"
            },
            "name": "q"
          }
        ]
      }
    }
  }
}
```

to allow arbitrary numbers of mask planes to be stored in memory efficiently, in groups of eight, without being constrained to jump from 32 mask planes directly to 64.

Additional components are represented by using Python Pydantic models[28] which provide validation and can define formal schemas. Every part of the model uses Pydantic and we intend to generate JSON Schema to allow formal validation by third parties. We have decided that where a model has an equivalent definition in the ASDF schema we will use a model that is compatible. We do not directly use the ASDF schema definitions but we do define ASDF-compatible variants for dealing with Astropy quantities, units, N-dimensional arrays, time, and table columns.

We have reimplemented the AFW component classes in Python, including bounded fields, aperture correction maps, and PSFs and these are used for the new `VisitImage` class. Additionally, we use a more general observation metadata model instead of `VisitInfo`.

In the initial set there is also a `ColorImage` class that consists of three images (red, green, and blue) and a `CellCoadd` that represents a modern implementation of our cell coadd types.[29,30] The system has been designed to allow flexibility in image classes but with a standardized serialization system.

5.2 Serialization System

A conscious design decision was to abstract the file output from the serialization infrastructure.

Every saveable Python type in the package (such as `Image`, `MaskedImage`, `VisitImage`, the various PSFs, transforms) has a paired serialization model — a Pydantic `ArchiveTree` subclass that mirrors the type's on-disk shape (e.g., `VisitImageSerializationModel` or `ColorImageSerializationModel`). The Python serialization model class is what is then persisted.

When a Python type is written, an `OutputArchive` for the backend is created and passed to that object's serialization method. A new archive tree Pydantic model is created where scalar fields are stored directly but large numerical arrays and tables are extracted and then referenced from the model. If an entity appears more than once in the model (such as a PSF) it is stored once and then referenced. Once the model is transformed to JSON the binary side channel information is then stored in a backend-specific way.

```
Filename: dp1.fits
No.    Name      Ver    Type      Cards   Dimensions   Format
  0  PRIMARY       1 PrimaryHDU     689   ()
  1  IMAGE         1 CompImageHDU    74   (4072, 4000)   float32
  2  MASK          1 CompImageHDU   109   (4072, 4000)   int32
  3  VARIANCE      1 CompImageHDU    74   (4072, 4000)   float32
  4  PSF/PIFF/INTERP/SOLUTION    1 BinTableHDU     12   1R x 1C   [9375D]
  5  JSON          1 BinTableHDU     11   1R x 1C   [PB(65720)]
  6  INDEX         1 BinTableHDU     23   6R x 7C   [24A, J, 8A, L, K, K, K]
```

Figure 2. A DP1 visit image converted to the new FITS serialization.

On read the JSON is located using the backend-specific code and validated. The model is then handed off to a deserialization method which resolves all the references to other parts of the file and reads those to reconstruct the Python type.

This design is what makes the four backends interchangeable. A Python class doesn't know whether it lives in a FITS HDU list or an HDF5 hierarchical structure; it only knows how to translate itself to and from a Pydantic tree that contains references. Each backend supplies its own `OutputArchive`/`InputArchive` (which knows where to put bytes) and its own referencing type (which knows how to spell a pointer to those bytes). Adding a fourth backend means writing those three pieces; no Python class needs to change. Validation, schema export, and version stamping all live on the shared `ArchiveTree` layer, applied uniformly across backends.

Listing 1 shows the JSON serialization of part of the PIFF model and how it refers to the location of the corresponding data array, which in this example is a path in the HDF5 hierarchy. This is tabular data and the serializer has the option of representing this as a table natively in the output file or instead representing it as separate columns that can be reconstructed on read into a table if the file format does not have native table support.

Details of the various back end implementations that we have developed are provided below. At this time the serializations other than FITS are exercises in demonstrating that we have included sufficient abstraction and flexibility in the data model; FITS is the reference data format for DP2. However, we may revisit this question for future data releases.

5.2.1 FITS Serialization

For FITS output we continue to write the image, mask, and variance to individual HDUs but the primary FITS header contains some additional keys to aid the deserialization tools. These headers are:

INDXADDR The byte offset to the start of the HDU containing the HDU index.

INDXSIZE The size of the index HDU in bytes.

JSONADDR The byte offset to the start of the HDU containing the JSON data model.

JSONSIZE The size of the JSON HDU in bytes.

At our scale, especially with the hybrid data center model we have adopted,[23] it is critical for performance to reduce the number of server round trips when performing a cutout[31] and the index allows us to efficiently skip to exactly the HDUs that we need to read without having to scan the entire file. We intend to submit the index HDU as a registered FITS convention once we can obtain some performance numbers. Nevertheless, in order to comply with the FITS standard, in addition to addressing FITS extensions via the byte-level offset information in the index HDU, we also ensure that each extension is identifiable with a unique (`EXTNAME`, `EXTVER`) pair.

For the documentation of the mask plane names and bit positions, the FITS header convention used in DP1, which originated from SDSS, has been replaced with one that avoids the use of the `HIERARCH` keyword and allows

for the inclusion of an explanatory phrase for each mask plane in addition to its symbolic name. We intend to submit this as a registered FITS convention as well.

As discussed above for DP1 data, although our WCS fits are not always representable as standard FITS-WCS headers, we always try to write a FITS WCS to the headers of each image HDU. For coadds this will be an accurate model, as they are in the exact TAN projection, but it will be an approximation for visit images. For LSSTCam visits the accuracy will be slightly worse at the edges of the focal plane than it was for the 9-detector LSSTComCam DP1 data. For DP2, at least, we will continue to use the SIP registered convention[16] to represent this approximation to the visit image astrometric distortions.

The `JSON` HDU contains the primary data model as JSON serialized from a Pydantic model. When the model contains large arrays of numbers those are pulled out of the JSON serialization and stored in FITS binary table or image HDUs. A reference to the FITS HDU is included in the JSON serialization so that on read the data can be located.

5.2.2 JSON Serialization

Pydantic models serialize to JSON as a native form and we have included support for a JSON serialization that also includes the numeric arrays. The main reason for doing this is to allow components (that might have a few thousand data values in an ancillary array) to be serialized with the Butler in a simpler text-based form, but the system is also able to serialize the full image types. For example, serializing a DP1 `VisitImage` to JSON results in a 600MB JSON file that can be compressed using Zstd to 200MB. We would not use full image JSON serialization in production but it does demonstrate the abstraction layer works.

5.2.3 HDF5 Serialization

To demonstrate that the serialization interface was flexible and not FITS-specific, we decided to also write an HDF5 plugin. We chose to use the Starlink N-Dimensional Data Format (NDF) data model[32] layered on the Starlink Hierarchical Data System (HDS) abstraction over HDF5.[33]

Our `MaskedImage` maps almost directly onto NDF: the image and variance arrays support pixel origin and units, and the WCS is represented by an AST native serialization. The model also has a convention for including direct support for FITS header cards, stored as an extension in a character array. There is one slight model mismatch concerning mask planes. NDF has a "quality" array corresponding to the mask but is limited to only 8 mask planes, whereas we can support an arbitrary number.

For the more complex data models NDF provides sufficient flexibility to be able to represent them, although there is no native support for tabular data. When there is additional information associated with a single image the convention is to place that in the extension "airlock" indicated by `.MORE`. This section can contain arbitrary additional structure and is where we write the JSON model. If there are multiple top-level images or masked images in the Python representation we would instead write those at the top level and the JSON component would also be at the top level since it is shared between all the images.

VisitImage Fig. 3 shows the NDF equivalent of the FITS file listed in Fig. 2. There is inherently more structure in the NDF representation. Here a `QUALITY` component is created from a collapsed version of the full 32-bit mask, to enable standard Starlink applications to view a form of the mask. The full mask is located at `MORE/LSST/MASK` and stores the 3-D version of the mask. The Pydantic model itself is located at `MORE/LSST/JSON` and internally it references explicit numerical components of the main file.

ColorImage A `ColorImage` is an image with WCS that represents three color channels. In the NDF serialization, shown in Fig. 4, each color channel is an NDF stored at the top level and the data model is now also stored at the top level in the `/LSST/JSON` character array rather than inside a particular image hierarchy. This demonstrates that we can use the hierarchy in multiple ways to group related concepts.

```
DATA_ARRAY     <ARRAY>          {structure}
   DATA(4072,4000)  <_REAL>        -6.884751,-6.867677,-6.850569, ... -5.527539,-5.532708
   ORIGIN(2)       <_INT64>        0,0

MORE           <EXT>            {structure}
   FITS(693)       <_CHAR*80>      'COMMENT   FITS (Flexible Image Transport System) format...'
                                   ... 'HIERARCH HAS-SIMULA...','EXPID   =          45508261376'
   LSST            <LSST>          {structure}
      JSON(1)         <_CHAR*88307>   '{"image":{"data":{"value":{"source":"ndf:/DATA_ARRAY/...'
      MASK            <NDF>           {structure}
         DATA_ARRAY     <ARRAY>         {structure}
            BAD_PIXEL       <_LOGICAL>      FALSE
            DATA(4072,4000,2)  <_UBYTE>    49,49,49,49,49,49,49,49, ... 0,0,0,0,0
            ORIGIN(3)       <_INT64>        0,0,0

         WCS             <WCS>           {structure}
            DATA(337)       <_CHAR*32>      ' Begin FrameSet',... ' End FrameSet'

      PSF             <PSF>           {structure}
         PIFF            <PIFF>          {structure}
            INTERP          <INTERP>        {structure}
               SOLUTION       <NDF>           {structure}
                  DATA_ARRAY     <ARRAY>         {structure}
                     DATA(15,625,1)  <_DOUBLE>      -8.80450720119882D-5, ... 4.17704930562368D-14
                     ORIGIN(3)      <_INT64>        0,0,0

QUALITY        <QUALITY>        {structure}
   BADBITS         <_UBYTE>        *
   QUALITY         <ARRAY>         {structure}
      BAD_PIXEL       <_LOGICAL>      FALSE
      DATA(4072,4000)  <_UBYTE>       1,1,1,1,1,1,1,1,1,1,1,1, ... 1,1,1,1,1,1,1
      ORIGIN(2)       <_INTEGER>      0,0

UNITS          <_CHAR*3>        'nJy'
VARIANCE       <ARRAY>          {structure}
   DATA(4072,4000)  <_REAL>        272.9544,272.951,272.9477, ... 259.7228,259.7221
   ORIGIN(2)       <_INT64>        0,0

WCS            <WCS>            {structure}
   DATA(312)       <_CHAR*32>      ' Begin FrameSet',... ' End FrameSet'
```

Figure 3. The structure of the HDF5 representation of a DP1 visit FITS file converted to use the NDF data model. This is the output from the Starlink `hdstrace` command, which uses HDF5 group attributes to match the Starlink HDS semantics.

```
BLUE           <NDF>           {structure}
   DATA_ARRAY     <ARRAY>         {structure}
      DATA(8,5)      <_UBYTE>        180,195,233,192,127,216,137,62,... 135,160,142,131
      ORIGIN(2)      <_INT64>        40,20

   WCS            <WCS>           {structure}
      DATA(187)      <_CHAR*32>      ' Begin FrameSet',... ' End FrameSet'

GREEN          <NDF>           {structure}
   DATA_ARRAY     <ARRAY>         {structure}
      DATA(8,5)      <_UBYTE>        209,3,200,106,54,88,31,99,143, ... 149,139,24
      ORIGIN(2)      <_INT64>        40,20

   WCS            <WCS>           {structure}
      DATA(187)      <_CHAR*32>      ' Begin FrameSet',... ' End FrameSet'

LSST           <LSST>           {structure}
   JSON(1)        <_CHAR*12703>   '{"red":{"data":{"source":"ndf:/RED/DATA_ARRAY/DATA","sha...'

RED            <NDF>           {structure}
   DATA_ARRAY     <ARRAY>         {structure}
      DATA(8,5)      <_UBYTE>        86,172,68,196,13,110,180,176,133,...129,88,115,175
      ORIGIN(2)      <_INT64>        40,20

   WCS            <WCS>           {structure}
      DATA(187)      <_CHAR*32>      ' Begin FrameSet',... ' End FrameSet'
```

Figure 4. The structure of the HDF5 representation of an RGB color image serialized with the NDF data model.

5.2.4 Zarr Serialization

The Zarr file format[†][34] is a cloud-optimized format designed for object stores. This is a relatively new format but it is being adopted by machine learning, bioimaging and geophysics communities.[35–37] The format is implemented as a hierarchical tree with metadata and compressed binary data kept as separate files and supporting chunking and sharding of large datasets to allow very efficient subsetting.

We have created an experimental Zarr writer that adopts standards from other scientific communities to try to maximize compatibility with existing tooling. We use the xarray / Climate and Forecasting conventions[‡] for labeling data arrays and defining masks and also OME-NGFF[36] conventions for multiscale support and simple affine transformation specifications where possible. By default we are using 256 × 256 pixel chunks with 16MB shards but we have not yet attempted to tune these parameters in a cloud environment.

In particular, we would like to consider whether our cutout service performance could be improved if we were to adopt Zarr instead of FITS,[38] although an issue we had not considered is that we use Rice lossy compression for our FITS files but Zarr has no registered equivalent. One obvious downside of that would be that we are required to serve the FITS files and would therefore have to increase the storage cost to also store some data products in Zarr format, this is a concern if the compression ratios are worse for Zarr. Furthermore, Zarr files are inherently directories with files inside and whilst switching to Zarr might lead to performance improvements in object stores for our data access patterns, it increases the burden on the archiving systems that now have to track many more small files. Our experimental Zarr bundles contain over 250 distinct files of various sizes, and performance critically depends on the number of files we need to access to do a single cutout. Given that each connection to the object store involves overhead we are concerned that even accessing a handful of files for a cutout will be slower than a single connection to a FITS file, especially given `fsspec` caching for related byte range requests that would not be relevant for separate files.

[†] https://zarr.dev
[‡] https://cfconventions.org

The JSON Pydantic model and the WCS are serialized as JSON using an LSST-specific naming convention since there is no existing standard for these data models. An example layout from a `VisitImage` can be found in listing 2 and the xarray interpretation of the same data file can be found in listing 3.

Listing 2. Overview of the Zarr layout for a `VisitImage`

```
/
|-- image (4000, 4072) float32
|-- lsst
|   |-- opaque_metadata
|   |   `-- fits
|   |       `-- primary (720, 80) uint8
|   `-- tables
|       `-- psf
|           `-- piff
|               `-- interp
|                   `-- solution
|                       `-- q (1, 625, 15) float64
|-- lsst_json (88349,) uint8
|-- mask (4000, 4072) uint16
`-- variance (4000, 4072) float32
```

Listing 3. Xarray interpretation of the same DP1 example Zarr file

```
<xarray.Dataset> Size: 163MB
Dimensions:    (y: 4000, x: 4072, None: 88349)
Dimensions without coordinates: y, x, None
Data variables:
    image      (y, x) float32 65MB dask.array<chunksize=(256, 256), meta=np.ndarray>
    lsst_json  (None) uint8 88kB dask.array<chunksize=(88349,), meta=np.ndarray>
    mask       (y, x) uint16 33MB dask.array<chunksize=(256, 256), meta=np.ndarray>
    variance   (y, x) float32 65MB dask.array<chunksize=(256, 256), meta=np.ndarray>
Attributes:
    data_model:  org.lsst.visit_image
    version:     1
    lsst:        {'version': 1, 'archive_class': 'VisitImage', 'json': 'lsst_...
    ome:         {'version': '0.5', 'multiscales': [{'name': 'visitimage', 'a...
```

6. FUTURE WORK

Now that we have demonstrated that the serialization system is general enough to support four output formats the next goal is to investigate other formats that have been mentioned by the community. Whilst it is not required that we support formats other than FITS, we feel that adding different formats and output data models can make the system more robust and expose bad assumptions.

ASDF,[24] which STScI are adopting for the Nancy Grace Roman Space Telescope[39] and which will drive shared analysis with Rubin LSST data, is being considered as an additional file format. We already share some common data models with ASDF, although we have no formal linkage to their schemas. ASDF uses a GWCS serialization[40] to represent world coordinates and the AST library does include support for using that model for export, although we might need to expand that support if there are mappings we use in AST that have no equivalent in GWCS. For the other data models we would have to investigate whether any of them warrant an attempt to expand the current formal ASDF schemas or whether we should treat them as non-standard extensions.

One further image type we intend to add is a class to represent multiple masked image cutouts. We will need something like this to represent outputs from our bulk cutout system[38] and using this serialization approach gives us some flexibility as to whether we serialize in the simpler "collection of images" way (such as using HDUs for each cutout, or the tabular "Green Bank convention") or a more compact approach favored by ML tooling

such as the Multimodal Universe[41] who recommend the use of N-dimensional arrays having all the cutout pixels in a single structure.

7. CONCLUSION

Our initial serialization system, developed in the early days of the LSST construction project,[42] successfully represented our C++ data models as FITS, but did not meet our cloud storage performance expectations or enable us to provide clear documentation to the community. We have developed a new approach that uses bespoke classes for different image types to replace an approach where a single class has to represent all possible metadata extensions. This has been coupled with an abstract file serialization system that lets us write the image data out in formats such as FITS and HDF5. To aid with cloud performance we are also proposing a new FITS convention whereby we store an index containing the byte offsets of each HDU. The intent is to switch to this new FITS format for Data Preview 2.

ACKNOWLEDGMENTS

This material is based upon work supported in part by the National Science Foundation through Cooperative Agreements AST-1258333 and AST-2241526 and Cooperative Support Agreements AST-1202910 and AST-2211468 managed by the Association of Universities for Research in Astronomy (AURA), and the Department of Energy under Contract No. DE-AC02-76SF00515 with the SLAC National Accelerator Laboratory managed by Stanford University. Additional Rubin Observatory funding comes from private donations, grants to universities, and in-kind support from LSST-DA Institutional Members. We thank Steve Pietrowicz for his review of the manuscript.

REFERENCES

[1] Ivezić, Ž., Kahn, S. M., Tyson, J. A., Abel, B., Acosta, E., Allsman, R., Alonso, D., AlSayyad, Y., Anderson, S. F., Andrew, J., Angel, J. R. P., Angeli, G. Z., Ansari, R., Antilogus, P., Araujo, C., Armstrong, R., Arndt, K. T., Astier, P., Aubourg, É., Auza, N., Axelrod, T. S., Bard, D. J., Barr, J. D., Barrau, A., Bartlett, J. G., Bauer, A. E., Bauman, B. J., Baumont, S., Bechtol, E., Bechtol, K., Becker, A. C., Becla, J., Beldica, C., Bellavia, S., Bianco, F. B., Biswas, R., Blanc, G., Blazek, J., Blandford, R. D., Bloom, J. S., Bogart, J., Bond, T. W., Booth, M. T., Borgland, A. W., Borne, K., Bosch, J. F., Boutigny, D., Brackett, C. A., Bradshaw, A., Brandt, W. N., Brown, M. E., Bullock, J. S., Burchat, P., Burke, D. L., Cagnoli, G., Calabrese, D., Callahan, S., Callen, A. L., Carlin, J. L., Carlson, E. L., Chandrasekharan, S., Charles-Emerson, G., Chesley, S., Cheu, E. C., Chiang, H.-F., Chiang, J., Chirino, C., Chow, D., Ciardi, D. R., Claver, C. F., Cohen-Tanugi, J., Cockrum, J. J., Coles, R., Connolly, A. J., Cook, K. H., Cooray, A., Covey, K. R., Cribbs, C., Cui, W., Cutri, R., Daly, P. N., Daniel, S. F., Daruich, F., Daubard, G., Daues, G., Dawson, W., Delgado, F., Dellapenna, A., de Peyster, R., de Val-Borro, M., Digel, S. W., Doherty, P., Dubois, R., Dubois-Felsmann, G. P., Durech, J., Economou, F., Eifler, T., Eracleous, M., Emmons, B. L., Fausti Neto, A., Ferguson, H., Figueroa, E., Fisher-Levine, M., Focke, W., Foss, M. D., Frank, J., Freemon, M. D., Gangler, E., Gawiser, E., Geary, J. C., Gee, P., Geha, M., Gessner, C. J. B., Gibson, R. R., Gilmore, D. K., Glanzman, T., Glick, W., Goldina, T., Goldstein, D. A., Goodenow, I., Graham, M. L., Gressler, W. J., Gris, P., Guy, L. P., Guyonnet, A., Haller, G., Harris, R., Hascall, P. A., Haupt, J., Hernandez, F., Herrmann, S., Hileman, E., Hoblitt, J., Hodgson, J. A., Hogan, C., Howard, J. D., Huang, D., Huffer, M. E., Ingraham, P., Innes, W. R., Jacoby, S. H., Jain, B., Jammes, F., Jee, M. J., Jenness, T., Jernigan, G., Jevremović, D., Johns, K., Johnson, A. S., Johnson, M. W. G., Jones, R. L., Juramy-Gilles, C., Jurić, M., Kalirai, J. S., Kallivayalil, N. J., Kalmbach, B., Kantor, J. P., Karst, P., Kasliwal, M. M., Kelly, H., Kessler, R., Kinnison, V., Kirkby, D., Knox, L., Kotov, I. V., Krabbendam, V. L., Krughoff, K. S., Kubánek, P., Kuczewski, J., Kulkarni, S., Ku, J., Kurita, N. R., Lage, C. S., Lambert, R., Lange, T., Langton, J. B., Le Guillou, L., Levine, D., Liang, M., Lim, K.-T., Lintott, C. J., Long, K. E., Lopez, M., Lotz, P. J., Lupton, R. H., Lust, N. B., MacArthur, L. A., Mahabal, A., Mandelbaum, R., Markiewicz, T. W., Marsh, D. S., Marshall, P. J., Marshall, S., May, M., McKercher, R., McQueen, M., Meyers, J., Migliore, M., Miller, M., and Mills, D. J., "LSST: From Science Drivers to Reference Design and Anticipated Data Products," *ApJ* **873**, 111 (March 2019). DOI: https://doi.org/10.3847/1538-4357/ab042c.

[2] Vera C. Rubin Observatory Team, Acero-Cuellar, T., Acosta, E., Adair, C. L., Adari, P., Adelman-McCarthy, J. K., Alexov, A., Allbery, R., Allsman, R., AlSayyad, Y., and et al., "The Vera C. Rubin Observatory Data Preview 1," *AJ* **171**, 360 (June 2026). DOI: https://doi.org/10.3847/1538-3881/ae521f.

[3] SLAC National Accelerator Laboratory and NSF-DOE Vera C. Rubin Observatory, "The LSST Camera (LSSTCam)," (2025). DOI: https://doi.org/10.71929/rubin/2571927.

[4] AlSayyad, Y. and O'Mullane, W., "Data Preview 2: Definition and planning," Technical Note RTN-111, NSF-DOE Vera C. Rubin Observatory (February 2026). https://rtn-111.lsst.io/.

[5] Aihara, H., AlSayyad, Y., Ando, M., Armstrong, R., Bosch, J., Egami, E., Furusawa, H., Furusawa, J., Harasawa, S., Harikane, Y., Hsieh, B.-C., Ikeda, H., Ito, K., Iwata, I., Kodama, T., Koike, M., Kokubo, M., Komiyama, Y., Li, X., Liang, Y., Lin, Y.-T., Lupton, R. H., Lust, N. B., MacArthur, L. A., Mawatari, K., Mineo, S., Miyatake, H., Miyazaki, S., More, S., Morishima, T., Murayama, H., Nakajima, K., Nakata, F., Nishizawa, A. J., Oguri, M., Okabe, N., Okura, Y., Ono, Y., Osato, K., Ouchi, M., Pan, Y.-C., Plazas Malagón, A. A., Price, P. A., Reed, S. L., Rykoff, E. S., Shibuya, T., Simunovic, M., Strauss, M. A., Sugimori, K., Suto, Y., Suzuki, N., Takada, M., Takagi, Y., Takata, T., Takita, S., Tanaka, M., Tang, S., Taranu, D. S., Terai, T., Toba, Y., Turner, E. L., Uchiyama, H., Vijarnwannaluk, B., Waters, C. Z., Yamada, Y., Yamamoto, N., and Yamashita, T., "Third data release of the Hyper Suprime-Cam Subaru Strategic Program," *PASJ* **74**, 247–272 (April 2022). DOI: https://doi.org/10.1093/pasj/psab122.

[6] FITS Working Group, "Definition of the Flexible Image Transport System (FITS)," (2018). https://fits.gsfc.nasa.gov/standard40/fits_standard40aa-le.pdf.

[7] Rubin Observatory Science Pipelines Developers, "The LSST Science Pipelines Software: Optical Survey Pipeline Reduction and Analysis Environment," Project Science Technical Note PSTN-019, NSF-DOE Vera C. Rubin Observatory (December 2025). DOI: https://doi.org/10.71929/rubin/2570545.

[8] Jenness, T., Bosch, J. F., Salnikov, A., Lust, N. B., Pease, N. M., Gower, M., Kowalik, M., Dubois-Felsmann, G. P., Mueller, F., and Schellart, P., "The Vera C. Rubin Observatory Data Butler and pipeline execution system," in [*Software and Cyberinfrastructure for Astronomy VII*], *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series* **12189**, 1218911 (August 2022). DOI: https://doi.org/10.1117/12.2629569.

[9] Aihara, H., Armstrong, R., Bickerton, S., Bosch, J., Coupon, J., Furusawa, H., Hayashi, Y., Ikeda, H., Kamata, Y., Karoji, H., Kawanomoto, S., Koike, M., Komiyama, Y., Lang, D., Lupton, R. H., Mineo, S., Miyatake, H., Miyazaki, S., Morokuma, T., Obuchi, Y., Oishi, Y., Okura, Y., Price, P. A., Takata, T., Tanaka, M. M., Tanaka, M., Tanaka, Y., Uchida, T., Uraguchi, F., Utsumi, Y., Wang, S.-Y., Yamada, Y., Yamanoi, H., Yasuda, N., Arimoto, N., Chiba, M., Finet, F., Fujimori, H., Fujimoto, S., Furusawa, J., Goto, T., Goulding, A., Gunn, J. E., Harikane, Y., Hattori, T., Hayashi, M., Hełminiak, K. G., Higuchi, R., Hikage, C., Ho, P. T. P., Hsieh, B.-C., Huang, K., Huang, S., Imanishi, M., Iwata, I., Jaelani, A. T., Jian, H.-Y., Kashikawa, N., Katayama, N., Kojima, T., Konno, A., Koshida, S., Kusakabe, H., Leauthaud, A., Lee, C.-H., Lin, L., Lin, Y.-T., Mandelbaum, R., Matsuoka, Y., Medezinski, E., Miyama, S., Momose, R., More, A., More, S., Mukae, S., Murata, R., Murayama, H., Nagao, T., Nakata, F., Niida, M., Niikura, H., Nishizawa, A. J., Oguri, M., Okabe, N., Ono, Y., Onodera, M., Onoue, M., Ouchi, M., Pyo, T.-S., Shibuya, T., Shimasaku, K., Simet, M., Speagle, J., Spergel, D. N., Strauss, M. A., Sugahara, Y., Sugiyama, N., Suto, Y., Suzuki, N., Tait, P. J., Takada, M., Terai, T., Toba, Y., Turner, E. L., Uchiyama, H., Umetsu, K., Urata, Y., Usuda, T., Yeh, S., and Yuma, S., "First data release of the Hyper Suprime-Cam Subaru Strategic Program," *PASJ* **70**, S8 (Jan. 2018). DOI: https://doi.org/10.1093/pasj/psx081.

[10] Bertin, E., "Automated Morphometry with SExtractor and PSFEx," in [*Astronomical Data Analysis Software and Systems XX*], Evans, I. N., Accomazzi, A., Mink, D. J., and Rots, A. H., eds., *Astronomical Society of the Pacific Conference Series* **442**, 435 (July 2011).

[11] Bertin, E., "PSFEx: Point Spread Function Extractor." Astrophysics Source Code Library, record ascl:1301.001 (January 2013).

[12] Jarvis, M., Meyers, J., Leget, P.-F., and Davis, C., "Piff: PSFs In the Full FOV." Astrophysics Source Code Library, record ascl:2102.024 (February 2021).

[13] Jarvis, M., Bernstein, G. M., Amon, A., Davis, C., Léget, P. F., Bechtol, K., Harrison, I., Gatti, M., Roodman, A., Chang, C., Chen, R., Choi, A., Desai, S., Drlica-Wagner, A., Gruen, D., Gruendl, R. A., Hernandez, A., MacCrann, N., Meyers, J., Navarro-Alsina, A., Pandey, S., Plazas, A. A., Secco, L. F.,

Sheldon, E., Troxel, M. A., Vorperian, S., Wei, K., Zuntz, J., Abbott, T. M. C., Aguena, M., Allam, S., Avila, S., Bhargava, S., Bridle, S. L., Brooks, D., Carnero Rosell, A., Carrasco Kind, M., Carretero, J., Costanzi, M., da Costa, L. N., De Vicente, J., Diehl, H. T., Doel, P., Everett, S., Flaugher, B., Fosalba, P., Frieman, J., García-Bellido, J., Gaztanaga, E., Gerdes, D. W., Gutierrez, G., Hinton, S. R., Hollowood, D. L., Honscheid, K., James, D. J., Kent, S., Kuehn, K., Kuropatkin, N., Lahav, O., Maia, M. A. G., March, M., Marshall, J. L., Melchior, P., Menanteau, F., Miquel, R., Ogando, R. L. C., Paz-Chinchón, F., Rykoff, E. S., Sanchez, E., Scarpine, V., Schubnell, M., Serrano, S., Sevilla-Noarbe, I., Smith, M., Suchyta, E., Swanson, M. E. C., Tarle, G., Varga, T. N., Walker, A. R., Wester, W., Wilkinson, R. D., and DES Collaboration, "Dark Energy Survey year 3 results: point spread function modelling," *MNRAS* **501**, 1282–1299 (Feb. 2021). DOI: https://doi.org/10.1093/mnras/staa3679.

[14] Berry, D. S., Warren-Smith, R. F., and Jenness, T., "AST: A library for modelling and manipulating coordinate systems," *Astronomy and Computing* **15**, 33–49 (April 2016). DOI: https://doi.org/10.1016/j.ascom.2016.02.003.

[15] Jenness, T. and Johnson, A. S., "Rubin Observatory Raw Data File Format," Camera Technical Note CTN-004, NSF-DOE Vera C. Rubin Observatory (January 2026). DOI: https://doi.org/10.71929/rubin/3011115.

[16] Shupe, D. L., Moshir, M., Li, J., Makovoz, D., Narron, R., and Hook, R. N., "The SIP Convention for Representing Distortion in FITS Image Headers," in [*Astronomical Data Analysis Software and Systems XIV*], Shopbell, P., Britton, M., and Ebert, R., eds., *Astronomical Society of the Pacific Conference Series* **347**, 491 (December 2005).

[17] Astropy Collaboration, Robitaille, T. P., Tollerud, E. J., Greenfield, P., Droettboom, M., Bray, E., Aldcroft, T., Davis, M., Ginsburg, A., Price-Whelan, A. M., Kerzendorf, W. E., Conley, A., Crighton, N., Barbary, K., Muna, D., Ferguson, H., Grollier, F., Parikh, M. M., Nair, P. H., Unther, H. M., Deil, C., Woillez, J., Conseil, S., Kramer, R., Turner, J. E. H., Singer, L., Fox, R., Weaver, B. A., Zabalza, V., Edwards, Z. I., Azalee Bostroem, K., Burke, D. J., Casey, A. R., Crawford, S. M., Dencheva, N., Ely, J., Jenness, T., Labrie, K., Lim, P. L., Pierfederici, F., Pontzen, A., Ptak, A., Refsdal, B., Servillat, M., and Streicher, O., "Astropy: A community Python package for astronomy," *A&A* **558**, A33 (October 2013). DOI: https://doi.org/10.1051/0004-6361/201322068.

[18] Wu, X., Roby, W., Goldian, T., Joliet, E., Ly, L., Mi, W., Wang, C., Zhang, L., Ciardi, D., and Dubois-Felsmann, G., "Next Generation Firefly for Web Application," in [*Astronomical Data Analysis Software and Systems XXVI*], Molinaro, M., Shortridge, K., and Pasian, F., eds., *Astronomical Society of the Pacific Conference Series* **521**, 32 (October 2019).

[19] Pitrou, A., "PEP 3154: Pickle Protocol, Version 4," Python Enhancement Proposal 3154, Python Software Foundation (2011). https://peps.python.org/pep-3154/.

[20] Pitrou, A., "PEP 574: Pickle Protocol 5 with Out-of-Band Data," Python Enhancement Proposal 574, Python Software Foundation (2018). https://peps.python.org/pep-0574/.

[21] Findeisen, K. and Bosch, J., "Improving Extensibility in afw.image.Exposure and Replacing afw.table.io," Data Management Technical Note DMTN-120, NSF-DOE Vera C. Rubin Observatory (March 2020). https://dmtn-120.lsst.io/.

[22] Bosch, J. F., "A New Approach to LSST's Image Data Models," Data Management Technical Note DMTN-251, NSF-DOE Vera C. Rubin Observatory (February 2026). https://dmtn-251.lsst.io/.

[23] O'Mullane, W., AlSayyad, Y., Chiang, J., Dubois, R., Economou, F., Hernandez, F., Huang, F., Jenness, T., Lim, K.-T., Li, Y.-T., Mueller, F., Speck, D., Pietrowicz, S., and Yang, W., "Rubin's hybrid on premises-cloud data access center," in [*Software and Cyberinfrastructure for Astronomy VIII*], Ibsen, J. and Chiozzi, G., eds., *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series* **13101**, 131012B (July 2024). DOI: https://doi.org/10.1117/12.3018005.

[24] Greenfield, P., Droettboom, M., and Bray, E., "ASDF: A new data format for astronomy," *Astronomy and Computing* **12**, 240–251 (Sept. 2015). DOI: https://doi.org/10.1016/j.ascom.2015.06.004.

[25] Chiappetti, L., Mink, J., Dobrzycki, A., and Taylor, M., "FITS and Data Representations WG BoF Session," in [*Astronomical Data Analysis Software and Systems XXVI*], Molinaro, M., Shortridge, K., and Pasian, F., eds., *Astronomical Society of the Pacific Conference Series* **521**, 733 (Oct. 2019).

[26] Shortridge, K., "Data Formats and Visualisation BoF," *arXiv e-prints* , arXiv:2605.20714 (May 2026). DOI: https://doi.org/10.48550/arXiv.2605.20714.

[27] Thomas, B., Jenness, T., Economou, F., Greenfield, P., Hirst, P., Berry, D. S., Bray, E., Gray, N., Muna, D., Turner, J., de Val-Borro, M., Santander-Vela, J., Shupe, D., Good, J., Berriman, G. B., Kitaeff, S., Fay, J., Laurino, O., Alexov, A., Landry, W., Masters, J., Brazier, A., Schaaf, R., Edwards, K., Redman, R. O., Marsh, T. R., Streicher, O., Norris, P., Pascual, S., Davie, M., Droettboom, M., Robitaille, T., Campana, R., Hagen, A., Hartogh, P., Klaes, D., Craig, M. W., and Homeier, D., "Learning from FITS: Limitations in use in modern astronomical research," *Astronomy and Computing* **12**, 133–145 (Sept. 2015). DOI: https://doi.org/10.1016/j.ascom.2015.01.009.

[28] Colvin, S., Jolibois, E., Ramezani, H., Garcia Badaracco, A., Dorsey, T., Montague, D., Matveenko, S., Trylesinski, M., Runkle, S., Hewitt, D., Hall, A., and Plot, V., "Pydantic," (June 2025). DOI: https://doi.org/10.5281/zenodo.15662245.

[29] Bosch, J. F., "File Formats and Layouts for Cell-based Coadds," Data Management Technical Note DMTN-294, NSF-DOE Vera C. Rubin Observatory (May 2026). https://dmtn-294.lsst.io/.

[30] Gorsuch, M. R. and Kannawadi, A., "Getting Started with Cell-Based Coadds," Commissioning Technical Note SITCOMTN-137, NSF-DOE Vera C. Rubin Observatory (May 2026). https://sitcomtn-137.lsst.io/.

[31] Allbery, R., "RSP image cutout service implementation strategy," Data Management Technical Note DMTN-208, NSF-DOE Vera C. Rubin Observatory (December 2024). https://dmtn-208.lsst.io/.

[32] Jenness, T., Berry, D. S., Currie, M. J., Draper, P. W., Economou, F., Gray, N., McIlwrath, B., Shortridge, K., Taylor, M. B., Wallace, P. T., and Warren-Smith, R. F., "Learning from 25 years of the extensible N-Dimensional Data Format," *Astronomy and Computing* **12**, 146–161 (Sept. 2015). DOI: https://doi.org/10.1016/j.ascom.2014.11.001.

[33] Jenness, T., "Reimplementing the Hierarchical Data System using HDF5," *Astronomy and Computing* **12**, 221–228 (Sept. 2015). DOI: https://doi.org/10.1016/j.ascom.2015.02.003.

[34] Miles, A., Bennett, D., jakirkham, Stansby, D., Orfanos, D. P., Hamman, J., Bussonnier, M., Moore, J., Jones, M., Augspurger, T., Cherian, D., Rzepka, N., Bourbeau, J., Verma, S., Spitz, H., Fulton, A., Abernathey, R., Lee, G., Gold, I., Kristensen, M. R. B., Patel, Z., Hunt-Isaak, I., Chopra, S., Rocklin, M., Zimmerman, N., AGHANGU, A. D., Chai, C. P., and de Andrade, E. S., "zarr-developers/zarr-python," (May 2026). DOI: https://doi.org/10.5281/zenodo.20038593.

[35] Gowan, T. A., Horel, J. D., Jacques, A. A., and Kovac, A., "Using cloud computing to analyze model output archived in zarr format," *Journal of Atmospheric and Oceanic Technology* **39**(4), 449 – 462 (2022). DOI: https://doi.org/10.1175/JTECH-D-21-0106.1.

[36] Moore, J., Basurto-Lozada, D., Besson, S., Bogovic, J., Bragantini, J., Brown, E. M., Burel, J.-M., Casas Moreno, X., de Medeiros, G., Diel, E. E., Gault, D., Ghosh, S. S., Gold, I., Halchenko, Y. O., Hartley, M., Horsfall, D., Keller, M. S., Kittisopikul, M., Kovacs, G., Küpcü Yoldaş, A., Kyoda, K., le Tournoulx de la Villegeorges, A., Li, T., Liberali, P., Lindner, D., Linkert, M., Lüthi, J., Maitin-Shepard, J., Manz, T., Marconato, L., McCormick, M., Lange, M., Mohamed, K., Moore, W., Norlin, N., Ouyang, W., Özdemir, B., Palla, G., Pape, C., Pelkmans, L., Pietzsch, T., Preibisch, S., Prete, M., Rzepka, N., Samee, S., Schaub, N., Sidky, H., Solak, A. C., Stirling, D. R., Striebel, J., Tischer, C., Toloudis, D., Virshup, I., Walczysko, P., Watson, A. M., Weisbart, E., Wong, F., Yamauchi, K. A., Bayraktar, O., Cimini, B. A., Gehlenborg, N., Haniffa, M., Hotaling, N., Onami, S., Royer, L. A., Saalfeld, S., Stegle, O., Theis, F. J., and Swedlow, J. R., "Ome-zarr: a cloud-optimized bioimaging file format with international community support," *Histochemistry and Cell Biology* **160**(3), 223–251 (2023). DOI: https://doi.org/10.1007/s00418-023-02209-1.

[37] Sansal, A., Kainkaryam, S., Lasscock, B., and Valenciano, A., "Mdio: Open-source format for multidimensional energy data," *The Leading Edge* **42**, 465–473 (07 2023). DOI: https://doi.org/10.1190/tle42070465.1.

[38] Jenness, T., Dubois-Felsmann, G. P., and Bosch, J. F., "Bulk Cutout Service Implementation Options," Data Management Technical Note DMTN-326, NSF-DOE Vera C. Rubin Observatory (May 2026). https://dmtn-326.lsst.io/.

[39] Greenfield, P., Slavich, E., Jamieson, W., and Dencheva, N., "The Advanced Scientific Data Format (ASDF): An Update," *SciPy 2022* (2022). DOI: https://doi.org/10.25080/majora-212e5952-000.

[40] Dencheva, N., Greenfield, P., and Droettboom, M., "GWCS: A Library for Managing World Coordinate Systems," in [*Astronomical Data Analysis Software and Systems XXV*], Lorente, N. P. F., Shortridge, K., and Wayth, R., eds., *Astronomical Society of the Pacific Conference Series* **512**, 609 (Dec. 2017).

[41] Angeloudi, E., Audenaert, J., Bowles, M., Boyd, B. M., Chemaly, D., Cherinka, B., Ciucă, I., Cranmer, M., Do, A., Grayling, M., Hayes, E. E., Hehir, T., Ho, S., Huertas-Company, M., Iyer, K. G., Jablonska, M., Lanusse, F., Leung, H. W., Mandel, K., Martínez-Galarza, J. R., Melchior, P., Meyer, L., Parker, L. H., Qu, H., Shen, J., Smith, M. J., Walmsley, M., Wu, J. F., and Multimodal Universe Collaboration, "The Multimodal Universe: 100 TB of Machine Learning Ready Astronomical Data," *Research Notes of the American Astronomical Society* **8**, 301 (Dec. 2024). DOI: https://doi.org/10.3847/2515-5172/ad9a63.

[42] Axelrod, T., Kantor, J., Lupton, R. H., and Pierfederici, F., "An open source application framework for astronomical imaging pipelines," in [*Software and Cyberinfrastructure for Astronomy*], Radziwill, N. M. and Bridger, A., eds., *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series* **7740**, 774015 (July 2010). DOI: https://doi.org/10.1117/12.857297.