The Advanced Scientific Data Format

Perry Greenfield

STScI

2019 – 3 – 21

Why ASDF?

- Primary driver: doing World Coordinate Systems right.
  - FITS fails miserably at this for most HST and JWST raw data
- More sensible and flexible organization of metadata and data than FITS.
  - FITS limitations are resulting in hard-to-maintain software.

Why not…

- VOTABLE
  - Can’t handle binary data efficiently
- HDF5
  - Complex, completely binary, and not self evident. Must use lengthy reference document to understand.
  - Only one really heavily used implementation.
    - Meaning that some bugs/variances from the standard may not be caught.
  - Poor support for dictionary structures.
  - Not as stable (questionable as an archival format)
  - Worries about split into paid and non-paid support communities

Summary of FITS Pros and Cons

- Pros
  - Metadata easy to read and transparent
  - Stable format (sort of…)
- Cons
  - Keyword/Value constraints extremely limiting
  - Limited organizational options
    - Not intrinsically hierarchical (conventions to overcome this are clumsy)
  - WCS standard mostly suitable for ideal projections but not much raw data
  - Standard and community makes enhancement extremely difficult
  - Community took lax view of non-conforming variants that libraries were forced to support. (Surprisingly true, to a lesser extent, for VOTABLE as well)

ASDF overview

- File consists of a YAML metadata header and optional binary data blocks.
  - Leverages a standard format and community-supported libraries.
  - YAML is a superset of JSON, entirely text format
  - Allows:
    - More formatting options
    - More concise and readable representations than JSON
    - Inline arrays and tables
    - References to one instance of a data structure can be shared eliminating the need for unnecessary duplication of information
- YAML header contains the metadata and all information on the organization of the data
- Versioning part of standard
- Permits schema definitions that can be used to check correctness of the YAML for defined objects
- Explicitly extensible. Anyone can define and publish their own schemas.
- Language independent. Currently only a full Python implementation but a C++ version is being developed.
- Not specific to Astronomy

GWCS

- Generalized World Coordinate System is a key feature for JWST supported in ASDF.
- Supports
  - arbitrarily complex coordinate transforms for multiple dimensions
  - arbitrary number of intermediate coordinate frames
  - References to output units and standard coordinate systems (e.g., FK5)
  - Multiple GWCS definitions (e.g., using different assumptions)
  - Parameterized input coordinates (e.g., spectral order, slit position, zero order position, etc)

STScI committed to long-term support of format

- Future improvements
  - Better ways to search and browse ASDF files for attributes or values
    - Both command line and GUI
  - C++ implementation
  - Generating prototype data model schemas for common telescope/instrument data.
  - Making it easier to work with (read and create) ASDF schemas
    - Provide more schema templates to mutate for new needs
    - Provide more concise and easy to read representations of schemas
      - They can be tedious to read and understand
- Planned use:
  - Used for GWCS within JWST data products (embedded in FITS extensions!)
  - Is an optional format for JWST data products (calibration pipelines support ASDF)
  - Being considered for WFIRST
  - Being used by DKIST

Links

- Paper: https://www.sciencedirect.com/science/article/pii/S2213133715000645?via%3Dihub
- Standard: https://asdf-standard.readthedocs.io/en/latest/
- Documentation: https://asdf.readthedocs.io/en/latest/
- Github: https://github.com/spacetelescope/asdf
- YAML: https://yaml.org/
- Schema: https://json-schema.org/understanding-json-schema/ (ASDF schemas are actually represented in YAML that maps directly to jsonschema)
- Use by machine learning team: https://blog.sourced.tech/post/asdf/

Developers

- Dan D’avella
- Mike Droettboom
- Erik Bray

Example

```
#ASDF 0.1.0
%YAML 1.1
%TAG ! tag:stsci.edu:asdf/0.1.0/
--- !core/asdf
data:
  $ref: "#/chips/0/science"
telescope: HST
instrument: ACS
proposal:
  id: 12699
  pi: Jane W. Astronomer
target:
  location:
    ra: 76.3775954471
    dec: 52.83079419491
    distance: .Inf
chips:
- chip_id: 1
  wcs: !wcs/wcs
      steps: !wcs/steps
        - !wcs/step
          name: detector
```

[rest of definition removed for brevity]

```
# what follows must adhere to core/asdf schema
```

- File starts with line indicating an ASDF file and version
- Lines that start before the yaml text with '%' are directives
  - %YAML 1.1 indicates yaml version
  - %TAG indicates the default schema environment (eliminates the need for a full url-like prefix on the tag)
- ! Indicates that the following is a tag and what follows it adheres to the tag's schema
- YAML text starts with '---' and ends with '…'
- $ indicates a reference to a label defined elsewhere
- Indented text indicates a block
- Blocks using 'key: value' sets are equivalent to Python dictionaries
- Blocks consisting of lines prefaced with '-' correspond to Python lists
- $ref is a reference to another item, in this case the 0 chip science array, so that the data attribute references that array (the chip_id: 1 entry at the bottom, continuing to next slide)

Example (cont.)

```
  science:                # science image;
    data: !core/ndarray  # references binary data block 2
      source: 2
      datatype: int32
      shape: [512, 512]
  data_quality:
    data: !core/ndarray
      source: 1
      datatype: int16
      shape: [512, 512]
  error:
    data: !core/ndarray
      source: 0
      datatype: float32
      shape: [512, 512]
- chip_id: 2
  wcs: !wcs/wcs # ... contents removed ...
  science:
    data: !core/ndarray
      source: 3
      datatype: float32
      shape: [512, 512]
  data_quality:  # ... contents removed ...
  error: # ... contents removed for brevity
```

WCS Example

```
#ASDF 0.1.0
%YAML 1.1
%TAG ! tag:stsci.edu:asdf/0.1.0/
--- !core/asdf  # the three dashes indicates start of yaml content
wcs: !wcs/wcs  # what follows must adhere to the wcs/wcs schema
  steps: !wcs/steps     # The beginning of the pipeline of transforms/frames of reference
    - !wcs/step         # The first transform/frame of reference (distortion correction)
      name: detector
      reference_position: [2048.0, 1024.0]
      axes:
        - !wcs/axis {type: detector, name: x, unit: !unit/unit pixel}
        - !wcs/axis {type: detector, name: y, unit: !unit/unit pixel}
      transform: !transform/concatenate
        forward:
          - !transform/polynomial
            coefficients:
              - [0.0, 0.0, 0.0, 0.0, 0.0]
              - [0.00077856419375, 0.1354312449693, 0.0, 0.0, 0.0]
              - [3.05198604166e-08, 3.3055309813e-06, -2.72696585312e-08, 0.0, 0.0]
              - [4.87638965665e-12, 3.86011101555e-12, 0.0, 0.0, 0.0]
              - [-2.680914674014611e-14, 0.0, 0.0, 0.0, 0.0]
            name: x_distortion_correction
          - !transform/polynomial
            coefficients:
              - [0.0, 0.0, 0.0, 0.0, 0.0]
              - [0.1209636405110, -0.0004185167199466, 0.0, 0.0, 0.0]
              - [3.620061079e-06, -3.0909053094e-08, 8.413534828e-07, 0.0, 0.0]
              - [-2.88214486e-11, 7.6239735e-12, 0.0, 0.0, 0.0]
              - [8.756611700935085e-14, 0.0, 0.0, 0.0, 0.0]
            name: y_distortion_correction
```

This distortion transform combines a 2-d distortion function for the corrected x coordinate and a 2-d distortion function for the corrected y coordinate

WCS Example (cont.)

```
  - !wcs/step # second transform/frame of reference
    name: focal_plane
    reference_position: [2048.0, 1024.0]
    axes:
      - !wcs/axis {type: focal_plane, name: x, unit: !unit/unit pixel}
      - !wcs/axis {type: focal_plane, name: y, unit: !unit/unit pixel}
    transform: !transform/compose
      forward:
        - !transform/concatenate
          forward:
            - !transform/shift {offset: 2048.0}
            - !transform/shift {offset: 1024.0}
        - !transform/affine
            matrix: !core/ndarray
              data:
              - [1.29058668e-05, 5.95320246e-06, 0.0]
              - [5.02215196e-06, -1.26450104e-05, 0.0]
              - [0.0, 0.0, 0.0]
              datatype: float64
              shape: [3, 3]
        - !transform/tangent {direction: forward}
        - !transform/rotate3d {
            direction: native2celestial, phi: 5.6305681099999996,
            psi: 180.0, theta: -72.054571839999994}
  - !wcs/step
    name: celestial
    axes:
      - !wcs/axis {type: celestial, name: RA, unit: !unit/unit deg}
      - !wcs/axis {type: celestial, name: DEC, unit: !unit/unit deg}
...
```

Defines transformation from focal plane to sky consisting of a shift, affine transformation, tangent projection and rotation

No transform; just defines output frame of reference