ASDF Standard

Release 1.0.0

Michael Droettboom, Erik Bray, et al
Space Telescope Science Institute

September 18, 2015

This document describes the Advanced Scientific Data Format (ASDF), pronounced *AZ*-diff.

This document is a work in progress and does not represent a released version of the ASDF standard.

CHAPTER 1

INTRODUCTION

The Flexible Image Transport System (FITS) has been the de facto standard for storing and exchanging astronomical data for decades, but it is beginning to show its age. Developed in the late 1970s, the FITS authors made a number of implementation choices that, while common at the time, are now seen to limit its utility for the needs of modern science. As astronomy moves into a more varied set of data product types (data models) with richer and more complex metadata, FITS is being pushed to its breaking point. The issues with FITS are outlined in great detail in *[Thomas2015]* (page 159).

Newer formats, such as VOTable (http://www.ivoa.net/documents/VOTable/) have partially addressed the problem of richer, more structured metadata, by using tree structures rather than flat key/value pairs. However, those text-based formats are unsuitable for storing large amounts of binary data. On the other end of the spectrum, formats such as HDF5 (http://www.hdfgroup.org/HDF5/) and BLZ (http://blaze.pydata.org/) address problems with large data sets and distributed computing, but don't really address the metadata needs of an interchange format. ASDF aims to exist in the same middle ground that made FITS so successful, by being a hybrid text and binary format: containing human editable metadata for interchange, and raw binary data that is fast to load and use. Unlike FITS, the metadata is highly structured and is designed up-front for extensibility.

ASDF has the following explicit goals:

- It has a hierarchical metadata structure, made up of basic dynamic data types such as strings, numbers, lists and mappings.
- It has human-readable metadata that can be edited directly in place in the file.
- The structure of the data can be automatically validated using schema.
- It's designed for extensibility: new conventions may be used without breaking backward compatibility with tools that do not understand those conventions. Versioning systems are used to prevent conflicting with alternative conventions.
- The binary array data (when compression is not used) is a raw memory dump, and techniques such as memory mapping can be used to efficiently access it.
- It is possible to read and write the file in as a stream, without requiring random access.
- It's built on top of industry standards, such as YAML (http://www.yaml.org) and JSON Schema (http://www.json-schema.org) to take advantage of a larger community working on the core problems of data representation. This also makes it easier to support ASDF in new programming languages and environments by building on top of existing libraries.
- Since every ASDF file has the version of the specification to which it is written, it will be possible, through careful planning, to evolve the ASDF format over time, allowing for files that use new features while retaining backward compatibility with older tools.

ASDF is primarily intended as an interchange format for delivering products from instruments to scientists or between scientists. While it is reasonably efficient to work with and transfer, it may not be optimal for direct use on large data sets in distributed and high performance computing environments. That is explicitly not a goal of the ASDF standard, as those requirements can sometimes be at odds with the needs of an interchange format.

ASDF still has a place in those environments as a delivery mechanism, even if it ultimately is not the actual format on which the computing is performed.

1.1 Implementations

The ASDF standard is being developed concurrently with a reference implementation written in Python (http://github.com/spacetelescope/pyasdf).

There is also a work-in-progress implementation for Go (http://github.com/astrogo/asdf) by Sebastian Binet.

1.2 Incorporated standards

The ASDF format is built on top of a number of existing standards:

- YAML 1.1 (http://yaml.org/spec/1.1/)
- JSON Schema Draft 4:
  - Core (http://tools.ietf.org/html/draft-zyp-json-schema-04)
  - Validation (http://tools.ietf.org/html/draft-fge-json-schema-validation-00)
  - Hyper-Schema (http://tools.ietf.org/html/draft-luff-json-hyper-schema-00)
- JSON Pointer (http://tools.ietf.org/html/rfc6901)
- Semantic Versioning 2.0.0 (http://semver.org/spec/v2.0.0.html)
- VOUnits (Units in the VO) (http://www.ivoa.net/documents/VOUnits/index.html)
- Zlib Deflate compression (http://www.zlib.net/feldspar.html)

CHAPTER 2

LOW-LEVEL FILE LAYOUT

The overall structure of a file is as follows (in order):

- *Header* (page 5)
- *Comments* (page 6), optional
- *Tree* (page 6), optional
- Zero or more *Blocks* (page 7)
- *Block index* (page 8), optional

ASDF is a hybrid text and binary format. The header, tree and block index are text, (specifically, in UTF-8 with DOS or UNIX-style newlines), while the blocks are raw binary.

The low-level file layout is designed in such a way that the tree section can be edited by hand, possibly changing its size, without requiring changes in other parts of the file. While such an operation may invalidate the *Block index* (page 8), the format is designed so that if the block index is removed or invalid, it may be regenerated by "skipping along" the blocks in the file.

The same is not true for resizing a block, which has an explicit size stored in the block header (except for, optionally, the last block).

Note also that, by design, an ASDF file containing no binary blocks is also a completely standard and valid YAML file.

Additionally, the spec allows for extra unallocated space after the tree and between blocks. This allows libraries to more easily update the files in place, since it allows expansion of certain areas without rewriting of the entire file.

2.1 Header

All ASDF files must start with a short one-line header. For example:

```
#ASDF 1.0.0
```

It is made up of two parts, separated by white space characters:

- **ASDF token**: The constant string `#ASDF`. This can be used to quickly identify the file as an ASDF file by reading the first 5 bytes. It begins with a # so it will be treated as a YAML comment such that the *Header* (page 5) and the *Tree* (page 6) together form a valid YAML file.
- **File format version**: The version of the low-level file format that this file was written with. This version may differ from the version of the ASDF specification, and is only updated when a change is made that affects the layout of file. It follows the Semantic Versioning 2.0.0 (http://semver.org/spec/v2.0.0.html) specification. See *Versioning Conventions* (page 15) for more information about these versions.

The header in EBNF form:

```
asdf_token = "#ASDF"
header     = asdf_token " " format_version ["\r"] "\n"
```

2.2 Comments

Additional comment lines may appear between the Header and the Tree.

The use of comments here is intended for information for the ASDF parser, and not information of general interest to the end user. All data of interest to the end user should be in the Tree.

Each line must begin with a # character.

2.3 Tree

The tree stores structured information using YAML Ain't Markup Language (YAML™) 1.1 (http://yaml.org/spec/1.1/) syntax. While it is the main part of most ASDF files, it is entirely optional, and a ASDF file may skip it completely. This is useful for creating files in *Exploded form* (page 9). Interpreting the contents of this section is described in greater detail in *The tree in-depth* (page 11). This section only deals with the serialized representation of the tree, not its logical contents.

The tree is always encoded in UTF-8, without an explicit byteorder marker (BOM). Newlines in the tree may be either DOS (`"\r\n"`) or UNIX (`"\n"`) format.

In ASDF 1.0.0, the tree must be encoded in YAML version 1.1 (http://yaml.org/spec/1.1/). At the time of this writing, the latest version of the YAML specification is 1.2, however most YAML parsers only support YAML 1.1, and the benefits of YAML 1.2 are minor. Therefore, for maximum portability, ASDF requires that the YAML is encoded in YAML 1.1. To declare that YAML 1.1 is being used, the tree must begin with the following line:

```
%YAML 1.1
```

The tree must contain exactly one YAML document, starting with `---` (YAML document start marker) and ending with `...` (YAML document end marker), each on their own line. Between these two markers is the YAML content. For example:

```
%YAML 1.1
%TAG ! tag:stsci.edu:asdf/
--- !core/asdf-1.0.0
data: !core/ndarray-1.0.0
  source: 0
  datatype: float64
  shape: [1024, 1024]
...
```

The size of the tree is not explicitly specified in the file, so that it can easily be edited by hand. Therefore, ASDF parsers must search for the end of the tree by looking for the end-of-document marker (`...`) on its own line. For example, the following regular expression may be used to find the end of the tree:

```
\r?\n...\r?\n
```

Though not required, the tree should be followed by some unused space to allow for the tree to be updated and increased in size without performing an insertion operation in the file. It also may be desirable to align the start of the first block to a filesystem block boundary. This empty space may be filled with any content (as long as it doesn't contain the `block_magic_token` described in *Blocks* (page 7)). It is recommended that the content is made up of space characters (`0x20`) so it appears as empty space when viewing the file.

2.4 Blocks

Following the tree and some empty space, or immediately following the header, there are zero or more binary blocks.

Blocks represent a contiguous chunk of binary data and nothing more. Information about how to interpret the block, such as the data type or array shape, is stored entirely in `ndarray` structures in the tree, as described in *ndarray* (page 20). This allows for a very flexible type system on top of a very simple approach to memory management within the file. It also allows for new extensions to ASDF that might interpret the raw binary data in ways that are yet to be defined.

There may be an arbitrary amount of unused space between the end of the tree and the first block. To find the beginning of the first block, ASDF parsers should search from the end of the tree for the first occurrence of the `block_magic_token`. If the file contains no tree, the first block must begin immediately after the header with no padding.

2.4.1 Block header

Each block begins with the following header:

- `block_magic_token` (4 bytes): Indicates the start of the block. This allows the file to contain some unused space in which to grow the tree, and to perform consistency checks when jumping from one block to the next. It is made up of the following 4 8-bit characters:
  - in hexadecimal: d3, 42, 4c, 4b
  - in ascii: `"\323BLK"`
- `header_size` (16-bit unsigned integer, big-endian): Indicates the size of the remainder of the header (not including the length of the `header_size` entry itself or the `block_magic_token`), in bytes. It is stored explicitly in the header itself so that the header may be enlarged in a future version of the ASDF standard while retaining backward compatibility. Importantly, ASDF parsers should not assume a fixed size of the header, but should obey the `header_size` defined in the file. In ASDF version 0.1, this should be at least 48, but may be larger, for example to align the beginning of the block content with a file system block boundary.
- `flags` (32-bit unsigned integer, big-endian): A bit field containing flags (described below).
- `compression` (4-byte byte string): The name of the compression algorithm, if any. Should be `\0\0\0\0` to indicate no compression. See *Compression* (page 8) for valid values.
- `allocated_size` (64-bit unsigned integer, big-endian): The amount of space allocated for the block (not including the header), in bytes.
- `used_size` (64-bit unsigned integer, big-endian): The amount of used space for the block on disk (not including the header), in bytes.
- `data_size` (64-bit unsigned integer, big-endian): The size of the block when decoded, in bytes. If `compression` is all zeros (indicating no compression), it **must** be equal to `used_size`. If compression is being used, this is the size of the decoded block data.
- `checksum` (16-byte string): An optional MD5 checksum of the used data in the block. The special value of all zeros indicates that no checksum verification should be performed.

2.4.2 Flags

The following bit flags are understood in the `flags` field:

- `STREAMED` (`0x1`): When set, the block is in streaming mode, and it extends to the end of the file. When set, the `allocated_size`, `used_size` and `data_size` fields are ignored. By necessity, any block with the `STREAMED` bit set must be the last block in the file.

2.4.3 Compression

Currently, two block compression types are supported:

- `zlib`: The zlib lossless compression algorithm. It is widely used, patent-unencumbered, and has an implementation released under a permissive license in zlib (http://www.zlib.net/).
- `bzp2`: The bzip2 lossless compression algorithm. It is widely used, assumed to be patent-unencumbered, and has an implementation released under a permissive license in the bzip2 library (http://www.bzip.org/).

2.4.4 Block content

Immediately following the block header, there are exactly `used_space` bytes of meaningful data, followed by `allocated_space - used_space` bytes of unused data. The exact content of the unused data is not enforced. The ability to have gaps of unused space allows an ASDF writer to reduce the number of disk operations when updating the file.

2.5 Block index

The block index allows for fast random access to each of the blocks in the file. It is completely optional: if not present, libraries may "skip along" the block headers to find the location of each block in the file. Libraries should detect invalid or obsolete block indices and ignore them and regenerate the index by skipping along the block headers.

The block index appears at the end of the file to make streaming an ASDF file possible without needing to determine the size of all blocks up front, which is non-trivial in the case of compression. It also allows for updating the index without an expensive insertion operation earlier in the file.

The block index must appear immediately after the allocated space for the last block in the file. If the last block is a streaming block, no block index may be present – the streaming block feature and block index are incompatible.

If no blocks are present in the file, the block index must also be absent.

The block index consists of a header, followed by a YAML document containing the indices of each block in the file.

The header must be exactly:

```
#ASDF BLOCK INDEX
```

followed by a DOS or UNIX newline.

Following the header is a YAML document (in YAML version 1.1, like the *Tree* (page 6)), containing a list of integers indicating the byte offset of each block in the file.

The following is an example block index:

```
#ASDF BLOCK INDEX
%YAML 1.1
--- [2043, 16340]
...
```

The offsets in the block index must be monotonically increasing, and must, by definition, be at least "block header size" apart. If they were allowed to appear in any order, it would be impossible to rebuild the index by skipping blocks were the index to become damaged or out-of-sync.

Additional zero-valued bytes may appear after the block index. This is mainly to support operating systems, such as Microsoft Windows, where truncating the file may not be easily possible.

2.5.1 Implementation recommendations

Libraries should look for the block index by reading backward from the end of the file.

Libraries should be conservative about what is an acceptable index, since addressing incorrect parts of the file could result in undefined behavior.

The following checks are recommended:

- Always ensure that the first offset entry matches the location of the first block in the file. This will catch the common use case where the YAML tree was edited by hand without updating the index. If they do not match, do not use the entire block index.
- Ensure that the last entry in the index refers to a block magic token, and that the end of the allocated space in the last block is immediately followed by the block index. If they do not match, do not use the entire block index.
- When using an offset in the block index, always ensure that the block magic token exists at that offset before reading data.

2.6 Exploded form

Exploded form expands a self-contained ASDF file into multiple files:

- An ASDF file containing only the header and tree, which by design is also a valid YAML file.
- *n* ASDF files, each containing a single block.

Exploded form is useful in the following scenarios:

- Not all text editors may handle the hybrid text and binary nature of the ASDF file, and therefore either can't open an ASDF file or would break an ASDF file upon saving. In this scenario, a user may explode the ASDF file, edit the YAML portion as a pure YAML file, and implode the parts back together.
- Over a network protocol, such as HTTP, a client may only need to access some of the blocks. While reading a subset of the file can be done using HTTP `Range` headers, not all web servers support this HTTP feature. Exploded form allows each block to be requested directly by a specific URI.
- An ASDF writer may stream a table to disk, when the size of the table is not known at the outset. Using exploded form simplifies this, since a standalone file containing a single table can be iteratively appended to without worrying about any blocks that may follow it.

Exploded form describes a convention for storing ASDF file content in multiple files, but it does not require any additions to the file format itself. There is nothing indicating that an ASDF file is in exploded form, other than the fact that some or all of its blocks come from external files. The exact way in which a file is exploded is up to the library and tools implementing the standard. In the simplest scenario, to explode a file, each *ndarray source property* (page 23) in the tree is converted from a local block reference into a relative URI.

CHAPTER 3

THE TREE IN-DEPTH

The ASDF tree, being encoded in YAML, is built out of the basic structures common to most dynamic languages: mappings (dictionaries), sequences (lists), and scalars (strings, integers, floating-point numbers, booleans, etc.). All of this comes "for free" by using YAML (http://yaml.org/spec/1.1/).

Since these core data structures on their own are so flexible, the ASDF standard includes a number of schema that define the structure of higher-level content. For instance, there is a schema that defines how *n-dimensional array data* (page 20) should be described. These schema are written in a language called *YAML Schema* (page 141) which is just a thin extension of JSON Schema, Draft 4 (http://json-schema.org/latest/json-schema-validation.html). (Such extensions are allowed and even encouraged by the JSON Schema standard, which defines the `$schema` attribute as a place to specify which extension is being used.) *ASDF schema definitions* (page 17), provides a reference to all of these schema in detail. *Extending ASDF* (page 141) describes how to use YAML schema to define new schema.

3.1 Tags

YAML includes the ability to assign *Tags* (page 11) (or types) to any object in the tree. This is an important feature that sets it apart from other data representation languages, such as JSON. ASDF defines a number of custom tags, each of which has a corresponding schema. For example the tag of the root element of the tree must always be `tag:stsci.edu:asdf/core/asdf-1.0.0`, which corresponds to the *asdf schema* (page 17) –in other words, the top level schema for ASDF trees. A validating ASDF reader would encounter the tag when reading in the file, load the corresponding schema, and validate the content against it. An ASDF library may also use this information to convert to a native data type that presents a more convenient interface to the user than the structure of basic types stored in the YAML content.

For example:

```
%YAML 1.1
--- !<tag:stsci.edu:asdf/core/asdf-1.0.0>
data: !<tag:stsci.edu:asdf/core/ndarray-1.0.0>
  source: 0
  datatype: float64
  shape: [1024, 1024]
  byteorder: little
...
```

All tags defined in the ASDF standard itself begin with the prefix `tag:stsci.edu:asdf/`. This can be broken down as:

- `tag:` The standard prefix used for all YAML tags.
- `stsci.edu` The owner of the tag.
- `asdf` The name of the standard.

Following that is the "module" containing the schema (see *ASDF schema definitions* (page 17) for a list of the available modules). Lastly is the tag name itself, for example, `asdf` or `ndarray`. Since it is cumbersome to type out these long prefixes for every tag, it is recommended that ASDF files declare a prefix at the top of the YAML file and use it throughout. (Most standard YAML writing libraries have facilities to do this automatically.) For example, the following example is equivalent to the above example, but is more user-friendly. The `%TAG` declaration declares that the exclamation point (`!`) will be replaced with the prefix `tag:stsci.edu:asdf/`:

```
%YAML 1.1
%TAG ! tag:stsci.edu:asdf/
--- !core/asdf-1.0.0
data: !core/ndarray-1.0.0
  source: 0
  datatype: float64
  shape: [1024, 1024]
  byteorder: little
```

An ASDF parser may use the tag to look up the corresponding schema in the ASDF standard and validate the element. The schema definitions ship as part of the ASDF standard.

An ASDF parser may also use the tag information to convert the element to a native data type. For example, in Python, an ASDF parser may convert a *ndarray* (page 20) tag to a Numpy (http://www.numpy.org) array instance, providing a convenient and familiar interface to the user to access *n*-dimensional data.

The ASDF standard does not require parser implementations to validate or perform native type conversion, however. A parser may simply leave the tree represented in the low-level basic data structures. When writing an ASDF file, however, the elements in the tree must be appropriately tagged for other tools to make use of them.

ASDF parsers must not fail when encountering an unknown tag, but must simply retain the low-level data structure and the presence of the tag. This is important, as end users will likely want to store their own custom tags in ASDF files alongside the tags defined in the ASDF standard itself, and the file must still be readable by ASDF parsers that do not understand those tags.

3.2 References

It is possible to directly reference other items within the same tree or within the tree of another ASDF file. This functionality is based on two IETF standards: JSON Pointer (IETF RFC 6901) (http://tools.ietf.org/html/rfc6901) and JSON Reference (Draft 3) (http://tools.ietf.org/html/draft-pbryan-zyp-json-ref-03).

A reference is represented as a mapping (dictionary) with a single key/value pair. The key is always the special keyword `$ref` and the value is a URI. The URI may contain a fragment (the part following the `#` character) in JSON Pointer syntax that references a specific element within the external file. This is a `/`-delimited path where each element is a mapping key or an array index. If no fragment is present, the reference refers to the top of the tree.

Note: JSON Pointer is a very simple convention. The only wrinkle is that because the characters `'~'` (0x7E) and `'/'` (0x2F) have special meanings, `'~'` needs to be encoded as `'~0'` and `'/'` needs to be encoded as `'~1'` when these characters appear in a reference token.

When these references are resolved, this mapping should be treated as having the same logical content as the target of the URI, though the exact details of how this is performed is dependent on the implementation, i.e., a library may copy the target data into the source tree, or it may insert a proxy object that is lazily loaded at a later time.

For example, suppose we had a given ASDF file containing some shared reference data, available on a public webserver at the URI `http://www.nowhere.com/reference.asdf`:

```
wavelengths:
  - !core/ndarray
    source: 0
    shape: [256, 256]
    datatype: float
    byteorder: little
```

Another file may reference this data directly:

```
reference_data:
  $ref: "http://www.nowhere.com/reference.asdf#wavelengths/0"
```

It is also possible to use references within the same file:

```
data: !core/ndarray
  source: 0
  shape: [256, 256]
  datatype: float
  byteorder: little
  mask:
    $ref: "#/my_mask"

my_mask: !core/ndarray
  source: 0
  shape: [256, 256]
  datatype: uint8
  byteorder: little
```

Reference resolution should be performed *after* the entire tree is read, therefore forward references within the same file are explicitly allowed.

Note: The YAML 1.1 standard itself also provides a method for internal references called "anchors" and "aliases". It does not, however, support external references. While ASDF does not explicitly disallow YAML anchors and aliases, since it explicitly supports all of YAML 1.1, their use is discouraged in favor of the more flexible JSON Pointer/JSON Reference standard described above.

3.3 Numeric literals

While it is possible to store arbitrary-sized integers as literals in YAML, not all programming languages and YAML libraries are able to read them. Therefore, to ensure portability, all numeric literals in the tree must assume that the reader has no more precision than that of a 64-bit double precision floating point number: 52-bits of precision. Therefore, ASDF libraries should refuse to write files containing integers that are larger than 52-bits.

3.4 Comments

It is quite common in FITS files to see comments that describe the purpose of the key/value pair. For example:

```
DATE    = '2015-02-12T23:08:51.191614' / Date this file was created (UTC)
TACID   = 'NOAO    '           / Time granting institution
```

Bringing this convention over to ASDF, one could imagine:

```
# Date this file was created (UTC)
creation_date: !time/utc
  2015-02-12T23:08:51.191614
# Time granting institution
time_granting_institution: NOAO
```

It should be obvious from the examples that these kinds of comments, describing the global meaning of a key, are much less necessary in ASDF. Since ASDF is not limited to 8-character keywords, the keywords themselves can be much more descriptive. But more importantly, the schema for a given key/value pair describes its purpose in detail. (It would be quite straightforward to build a tool that, given an entry in a YAML tree, looks up the schema's description associated with that entry.) Therefore, the use of comments to describe the global meaning of a value are strongly discouraged.

However, there still may be cases where a comment may be desired in ASDF, such as when a particular value is unusual or unexpected. The YAML standard includes a convention for comments, providing a handy way to include annotations in the ASDF file:

```
# We set this to filter B here, even though C is the more obvious
# choice, because B is handled with more accuracy by our software.
filter:
  type: B
```

Unfortunately, most YAML parsers will simply throw these comments out and do not provide any mechanism to retain them, so reading in an ASDF file, making some changes, and writing it out will remove all comments. Even if the YAML parser could be improved or extended to retain comments, the YAML standard does not define which values the comments are associated with. In the above example, it is only by standard reading conventions that we assume the comment is associated with the content following it. If we were to move the content, where should the comment go?

To provide a mechanism to add user comments without swimming upstream against the YAML standard, we recommend a convention for associating comments with objects (mappings) by using the reserved key name `//`. In this case, the above example would be rewritten as:

```
filter:
  //: |
    We set this to filter B here, even though C was used, because B
    is handled with more accuracy by our software.
  type: B
```

ASDF parsers must not interpret or react programmatically to these comment values: they are for human reference only. No schema may use `//` as a meaningful key.

CHAPTER 4

VERSIONING CONVENTIONS

One of the explicit goals of ASDF is to be as future proof as possible. This involves being able to add features as needed while still allowing older libraries that may not understand those new features to reasonably make sense of the rest of the file.

The ASDF standard includes three categories of versions, all of which may advance independently of one another.

- **Standard version**: The version of the standard as a whole. This version provides a convenient handle to refer to a particular snapshot of the ASDF standard at a given time. This allows libraries to advertise support for "ASDF standard version X.Y.Z".
- **File format version**: Refers to the version of the blocking scheme and other details of the low-level file layout. This is the number that appears on the `#ASDF` header line at the start of every ASDF file and is essential to correctly interpreting the various parts of an ASDF file.
- **Schema versions**: Each schema for a particular YAML tag is individually versioned. This allows schemas to evolve, while still allowing data written to an older version of the schema to be validated correctly.

  Schemas provided by third parties (i.e. not in the ASDF specification itself) are also strongly encouraged to be versioned as well.

Version numbers all follow the same convention according to the Semantic Versioning 2.0.0 (http://semver.org/spec/v2.0.0.html) specification.

- **major version**: The major version number advances when a backward incompatible change is made. For example, this would happen when an existing property in a schema changes meaning. (An exception to this is that when the major version is 0, there are no guarantees of backward compatibility.)
- **minor version**: The minor version number advances when a backward compatible change is made. For example, this would happen when new properties are added to a schema.
- **patch version**: The patch version number advances when a minor change is made that does not directly affect the file format itself. For example, this would happen when a misspelling or grammatical error in the specification text is made that does not affect the interpretation of an ASDF file.
- **pre-release version**: An optional fourth part may also be present following a hyphen to indicate a pre-release version in development. For example, the pre-release of version `1.2.3` would be `1.2.3-dev+a2c4`.

4.1 Relationship of version numbers

The major number in the **standard version** is incremented whenever the major number in the **file format version** is incremented.

At present the **schema versions** move in lock-step with the **standard version**. However, in the future, we may break from that convention, so libraries should address versions of individual schemas independently.

4.2 Handling version mismatches

Given these conventions, the ASDF standard recommends certain behavior of ASDF libraries. ASDF libraries should, but are not required, to support as many existing versions of the file format and schemas as possible, and use the version numbers in the file to act accordingly.

For future-proofing, the library should gracefully handle version numbers that are greater than those understood by the library. The following applies to both kinds of version numbers that appear in the file: the **file format version** and **schema versions**.

- When encountering a **major version** that is greater than the understood version, by default, an exception should be raised. This behavior may be overridden through explicit user interaction, in which case the library will attempt to handle the element using the conventions of the most recent understood version.
- When encountering a **minor version** that is greater than the understood version, a warning should be emitted, and the library should attempt to handle the element using the conventions of the most recent understood version.
- When encountering a **patch version** that is greater than the understood version, silently ignore the difference and handle the element using the conventions of the most recent understood version.

When writing ASDF files, it is recommended that libraries provide both of the following modes of operation:

- Upgrade the file to the latest versions of the file format and schemas understood by the library.
- Preserve the version of the ASDF standard used by the input file.

Writing out a file that mixes versions of schema from different versions of the ASDF standard is not recommended, though such a file should be accepted by readers given the rules above.

CHAPTER 5

ASDF SCHEMA DEFINITIONS

This reference section describes the schema files for the built-in tags in ASDF.

ASDF schemas are arranged into "modules". All ASDF implementations must support the "core" module, but the other modules are optional.

5.1 Core

The `core` module contains schema that must be implemented by every asdf library.

5.1.1 asdf: Top-level schema for every ASDF file.

Type: object.

Top-level schema for every ASDF file.

This schema contains the top-level attributes for every ASDF file.

Properties:

`asdf_library`

Type: software-1.0.0 (page 39).

Describes the ASDF library that produced the file.

`history`

Type: array of (history_entry-1.0.0 (page 40)).

A log of transformations that have happened to the file. May include such things as data collection, data calibration pipelines, data analysis etc.

Items:

Type: history_entry-1.0.0 (page 40).

`data`

Type: ndarray-1.0.0 (page 20).

The data array corresponds to the main science data array in the file. Oftentimes, the data model will be much more complex than a single array, but this array will be used by applications that just want to convert to a display an image or preview of the file. It is recommended, but not required, that it is a 2-dimensional image array.

`fits`

Type: fits-1.0.0 (page 41).

A way to specify exactly how this ASDF file should be converted to FITS.

wcs

Type: wcs-1.0.0 (page 125).

The location of the main WCS for the main data.

Original schema in YAML

```yaml
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/core/asdf-1.0.0"
title: |
  Top-level schema for every ASDF file.

description: |
  This schema contains the top-level attributes for every ASDF file.

tag: "tag:stsci.edu:asdf/core/asdf-1.0.0"
type: object
properties:
  asdf_library:
    description: |
      Describes the ASDF library that produced the file.
    $ref: "software-1.0.0"

  history:
    description: |
      A log of transformations that have happened to the file.  May
      include such things as data collection, data calibration
      pipelines, data analysis etc.
    type: array
    items:
      $ref: "history_entry-1.0.0"

  data:
    description: |
      The data array corresponds to the main science data array in the
      file.  Oftentimes, the data model will be much more complex than
      a single array, but this array will be used by applications that
      just want to convert to a display an image or preview of the
      file.  It is recommended, but not required, that it is a
      2-dimensional image array.
    $ref: "ndarray-1.0.0"

  fits:
    description: |
      A way to specify exactly how this ASDF file should be converted
      to FITS.
    $ref: "../fits/fits-1.0.0"

  wcs:
    description: |
      The location of the main WCS for the main data.
    $ref: "../wcs/wcs-1.0.0"
```

```
additionalProperties: true
```

5.1.2 complex: Complex number value.

Type: string (regex `([-+]?[0-9]*\.?[0-9]+([eE][-+]?[0-9]+)?)?([-+][0-9]*\.?[0-9]+([eE][-+]?[0-9]+)?[JjIi])?`).

Complex number value.

Represents a complex number matching the following EBNF grammar

```
plus-or-minus = "+" | "-"
suffix        = "J" | "j" | "I" | "i"
complex       = [ieee754] [plus-or-minus ieee754 suffix]
```

Where `ieee754` is a floating point number in IEEE 754 decimal format.

Though `J`, `j`, `I` and `i` must be supported on reading, it is recommended to use `i` on writing.

Examples:

1 real, -1 imaginary:

```
!core/complex-1.0.0 1-1j
```

0 real, 1 imaginary:

```
!core/complex-1.0.0 1J
```

-1 real, 0 imaginary:

```
!core/complex-1.0.0 -1
```

Original schema in YAML

```
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/core/complex-1.0.0"
title: Complex number value.
description: |
  Represents a complex number matching the following EBNF grammar

  ```
    plus-or-minus = "+" | "-"
    suffix        = "J" | "j" | "I" | "i"
    complex       = [ieee754] [plus-or-minus ieee754 suffix]
  ```

  Where `ieee754` is a floating point number in IEEE 754 decimal
  format.

  Though `J`, `j`, `I` and `i` must be supported on reading, it is
  recommended to use `i` on writing.

```

```
examples:
  -
    - 1 real, -1 imaginary
    - "!core/complex-1.0.0 1-1j"
  -
    - 0 real, 1 imaginary
    - "!core/complex-1.0.0 1J"
  -
    - -1 real, 0 imaginary
    - "!core/complex-1.0.0 -1"

tag: "tag:stsci.edu:asdf/core/complex-1.0.0"
type: string
pattern: "([-+]?[0-9]*\\.?[0-9]+([eE][-+]?[0-9]+)?)?([-+][0-9]*\\.?[0-9]+([eE][-+]?[0-9]+)?[JjIi])?"
```

5.1.3 ndarray: An *n*-dimensional array.

Type: *definitions/inline-data* (page 22) or object.

An n-dimensional array.

There are two ways to store the data in an ndarray.

- Inline in the tree: This is recommended only for small arrays. In this case, the entire `ndarray` tag may be a nested list, in which case the type of the array is inferred from the content. (See the rules for type inference in the `inline-data` definition below.) The inline data may also be given in the `data` property, in which case it is possible to explicitly specify the `datatype` and other properties.
- External to the tree: The data comes from a *block* (page 7) within the same ASDF file or an external ASDF file referenced by a URI.

Definitions:

`scalar-datatype`

Type: string from [”int8”, “uint8”, “int16”, “uint16”, “int32”, “uint32”, “int64”, “uint64”, “float32”, “float64”, “complex64”, “complex128”, “bool8”] or array.

Describes the type of a single element.

There is a set of numeric types, each with a single identifier:

- `int8`, `int16`, `int32`, `int64`: Signed integer types, with the given bit size.
- `uint8`, `uint16`, `uint32`, `uint64`: Unsigned integer types, with the given bit size.
- `float32`: Single-precision floating-point type or “binary32”, as defined in IEEE 754.
- `float64`: Double-precision floating-point type or “binary64”, as defined in IEEE 754.
- `complex64`: Complex number where the real and imaginary parts are each single-precision floating-point (“binary32”) numbers, as defined in IEEE 754.
- `complex128`: Complex number where the real and imaginary parts are each double-precision floating-point (“binary64”) numbers, as defined in IEEE 754.

There are two distinct fixed-length string types, which must be indicated with a 2-element array where the first element is an identifier for the string type, and the second is a length:

- `ascii`: A string containing ASCII text (all codepoints < 128), where each character is 1 byte.

- `ucs4`: A string containing unicode text in the UCS-4 encoding, where each character is always 4 bytes long. Here the number of bytes used is 4 times the given length.

Any of:

Type: string from [”int8”, “uint8”, “int16”, “uint16”, “int32”, “uint32”, “int64”, “uint64”, “float32”, “float64”, “complex64”, “complex128”, “bool8”].

Type: array.

Items:

`index[0]`

Type: string from [”ascii”, “ucs4”].

`index[1]`

Type: integer ≥ 0.

`datatype`

Type: *definitions/scalar-datatype* (page 20) or array of (*definitions/scalar-datatype* (page 20) or object).

The data format of the array elements. May be a single scalar datatype, or may be a nested list of datatypes. When a list, each field may have a name.

Any of:

Type: *definitions/scalar-datatype* (page 20).

Type: array of (*definitions/scalar-datatype* (page 20) or object).

Items:

Type: *definitions/scalar-datatype* (page 20) or object.

Any of:

Type: *definitions/scalar-datatype* (page 20).

Type: object.

Properties:

`name`

Type: string (regex `[A-Za-z_][A-Za-z0-9_]*`).

The name of the field

`datatype`

Type: *definitions/datatype* (page 21). Required.

`byteorder`

Type: string from [”big”, “little”].

The byteorder for the field. If not provided, the byteorder of the datatype as a whole will be used.

`shape`

Type: array of (integer ≥ 0).

Items:

Type: integer ≥ 0.

`inline-data`

Type: array of (number or string or null or complex-1.0.0 (page 19) or *definitions/inline-data* (page 22) or boolean).

Inline data is stored in YAML format directly in the tree, rather than referencing a binary block. It is made out of nested lists.

If the datatype of the array is not specified, it is inferred from the array contents. Type inference is supported only for homogeneous arrays, not tables.

- If any of the elements in the array are YAML strings, the `datatype` of the entire array is `ucs4`, with the width of the largest string in the column, otherwise...
- If any of the elements in the array are complex numbers, the `datatype` of the entire column is `complex128`, otherwise...
- If any of the types in the column are numbers with a decimal point, the `datatype` of the entire column is `float64`, otherwise..
- If any of the types in the column are integers, the `datatype` of the entire column is `int64`, otherwise...
- The `datatype` of the entire column is `bool8`.

Masked values may be included in the array using `null`. If an explicit mask array is also provided, it takes precedence.

Items:

Type: number or string or null or complex-1.0.0 (page 19) or *definitions/inline-data* (page 22) or boolean.

Any of:

Type: number.

Type: string.

Type: null.

Type: complex-1.0.0 (page 19).

Type: *definitions/inline-data* (page 22).

Type: boolean.

Any of:

Type: *definitions/inline-data* (page 22).

Type: object.

Properties:

`source`

Type: integer or string (format uri).

The source of the data.

- If an integer: If positive, the zero-based index of the block within the same file. If negative, the index from the last block within the same file. For example, a source of `-1` corresponds to the last block in the same file.
- If a string, a URI to an external ASDF file containing the block data. Relative URIs and `file:` and `http:` protocols must be supported. Other protocols may be supported by specific library implementations.

The ability to reference block data in an external ASDF file is intentionally limited to the first block in the external ASDF file, and is intended only to support the needs of *exploded* (page 9). For the more general case of referencing data in an external ASDF file, use tree *references* (page 12).

Any of:

Type: integer.

Type: string (format uri).

`data`

Type: *definitions/inline-data* (page 22).

The data for the array inline.

If `datatype` and/or `shape` are also provided, they must match the data here and can be used as a consistency check. `strides`, `offset` and `byteorder` are meaningless when `data` is provided.

`shape`

Type: array of (integer $\geq$ 0 or any from ["*"]).

The shape of the array.

The first entry may be the string `*`, indicating that the length of the first index of the array will be automatically determined from the size of the block. This is used for streaming support.

Items:

Type: integer $\geq$ 0 or any from ["*"].

Any of:

Type: integer $\geq$ 0.

Type: any from ["*"].

`datatype`

Type: *definitions/datatype* (page 21).

The data format of the array elements.

`byteorder`

Type: string from [”big”, “little”].

The byte order (big- or little-endian) of the array data.

`offset`

Type: integer ≥ 0.

The offset, in bytes, within the data for this start of this view.

Default: 0

`strides`

Type: array of (integer ≥ 1 or integer ≤ -1).

The number of bytes to skip in each dimension. If not provided, the array is assumed by be contiguous and in C order. If provided, must be the same length as the shape property.

Items:

Type: integer ≥ 1 or integer ≤ -1.

Any of:

`---`

Type: integer ≥ 1.

`---`

Type: integer ≤ -1.

`mask`

Type: number or complex-1.0.0 (page 19) or ndarray-1.0.0 (page 20) and any.

Describes how missing values in the array are stored. If a scalar number, that number is used to represent missing values. If an ndarray, the given array provides a mask, where non-zero values represent missing values in this array. The mask array must be broadcastable to the dimensions of this array.

Any of:

`---`

Type: number.

`---`

Type: complex-1.0.0 (page 19).

`---`

Type: ndarray-1.0.0 (page 20) and any.

All of:

`0`

Type: ndarray-1.0.0 (page 20).

`1`

Type: any.

Examples:

An inline array, with implicit data type:

```
!core/ndarray-1.0.0
  [[1, 0, 0],
   [0, 1, 0],
   [0, 0, 1]]
```

An inline array, with an explicit data type:

```
!core/ndarray-1.0.0
  datatype: float64
  data:
    [[1, 0, 0],
     [0, 1, 0],
     [0, 0, 1]]
```

An inline structured array, where the types of each column are automatically detected:

```
!core/ndarray-1.0.0
  [[M110, 110, 205, And],
   [ M31,  31, 224, And],
   [ M32,  32, 221, And],
   [M103, 103, 581, Cas]]
```

An inline structured array, where the types of each column are explicitly specified:

```
!core/ndarray-1.0.0
  datatype: [['ascii', 4], uint16, uint16, ['ascii', 4]]
  data:
    [[M110, 110, 205, And],
     [ M31,  31, 224, And],
     [ M32,  32, 221, And],
     [M103, 103, 581, Cas]]
```

A double-precision array, in contiguous memory in a block within the same file:

```
!core/ndarray-1.0.0
  source: 0
  shape: [1024, 1024]
  datatype: float64
  byteorder: little
```

A view of a tile in that image:

```
!core/ndarray-1.0.0
  source: 0
  shape: [256, 256]
  datatype: float64
  byteorder: little
  strides: [8192, 8]
  offset: 2099200
```

A structured datatype, with nested columns for a coordinate in (*ra*, *dec*), and a 3x3 convolution kernel:

```
!core/ndarray-1.0.0
  source: 0
  shape: [64]
  datatype:
    - name: coordinate
      datatype:
        - name: ra
          datatype: float64
        - name: dec
          datatype: float64
    - name: kernel
      datatype: float32
      shape: [3, 3]
  byteorder: little
```

An array in Fortran order:

```
!core/ndarray-1.0.0
  source: 0
  shape: [1024, 1024]
  datatype: float64
  byteorder: little
  strides: [8192, 8]
```

An array where values of -999 are treated as missing:

```
!core/ndarray-1.0.0
  source: 0
  shape: [256, 256]
  datatype: float64
  byteorder: little
  mask: -999
```

An array where another array is used as a mask:

```
!core/ndarray-1.0.0
  source: 0
  shape: [256, 256]
  datatype: float64
  byteorder: little
  mask: !core/ndarray-1.0.0
    source: 1
    shape: [256, 256]
    datatype: bool8
    byteorder: little
```

An array where the data is stored in the first block in another ASDF file.:

```
!core/ndarray-1.0.0
  source: external.asdf
  shape: [256, 256]
  datatype: float64
  byteorder: little
```

Original schema in YAML

```yaml
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/asdf/asdf-schema-1.0.0"
id: "http://stsci.edu/schemas/asdf/core/ndarray-1.0.0"
tag: "tag:stsci.edu:asdf/core/ndarray-1.0.0"

title: >
  An *n*-dimensional array.

description: |
  There are two ways to store the data in an ndarray.

  - Inline in the tree: This is recommended only for small arrays.  In
    this case, the entire ``ndarray`` tag may be a nested list, in
    which case the type of the array is inferred from the content.
    (See the rules for type inference in the ``inline-data``
    definition below.)  The inline data may also be given in the
    ``data`` property, in which case it is possible to explicitly
    specify the ``datatype`` and other properties.

  - External to the tree: The data comes from a [block](ref:block)
    within the same ASDF file or an external ASDF file referenced by a
    URI.

examples:
  -
    - An inline array, with implicit data type
    - |
        !core/ndarray-1.0.0
          [[1, 0, 0],
           [0, 1, 0],
           [0, 0, 1]]

  -
    - An inline array, with an explicit data type
    - |
        !core/ndarray-1.0.0
          datatype: float64
          data:
            [[1, 0, 0],
             [0, 1, 0],
             [0, 0, 1]]

  -
    - An inline structured array, where the types of each column are
      automatically detected
    - |
        !core/ndarray-1.0.0
          [[M110, 110, 205, And],
           [ M31,  31, 224, And],
           [ M32,  32, 221, And],
           [M103, 103, 581, Cas]]

  -
    - An inline structured array, where the types of each column are
      explicitly specified
```

```
    - |
        !core/ndarray-1.0.0
          datatype: [['ascii', 4], uint16, uint16, ['ascii', 4]]
          data:
            [[M110, 110, 205, And],
             [ M31,  31, 224, And],
             [ M32,  32, 221, And],
             [M103, 103, 581, Cas]]

  -
    - A double-precision array, in contiguous memory in a block within
      the same file
    - |
        !core/ndarray-1.0.0
          source: 0
          shape: [1024, 1024]
          datatype: float64
          byteorder: little

  -
    - A view of a tile in that image
    - |
        !core/ndarray-1.0.0
          source: 0
          shape: [256, 256]
          datatype: float64
          byteorder: little
          strides: [8192, 8]
          offset: 2099200

  -
    - A structured datatype, with nested columns for a coordinate in
      (*ra*, *dec*), and a 3x3 convolution kernel
    - |
        !core/ndarray-1.0.0
          source: 0
          shape: [64]
          datatype:
            - name: coordinate
              datatype:
                - name: ra
                  datatype: float64
                - name: dec
                  datatype: float64
            - name: kernel
              datatype: float32
              shape: [3, 3]
          byteorder: little

  -
    - An array in Fortran order
    - |
        !core/ndarray-1.0.0
          source: 0
          shape: [1024, 1024]
          datatype: float64
          byteorder: little
          strides: [8192, 8]
```

```
  -
    - An array where values of -999 are treated as missing
    - |
        !core/ndarray-1.0.0
          source: 0
          shape: [256, 256]
          datatype: float64
          byteorder: little
          mask: -999

  -
    - An array where another array is used as a mask
    - |
        !core/ndarray-1.0.0
          source: 0
          shape: [256, 256]
          datatype: float64
          byteorder: little
          mask: !core/ndarray-1.0.0
            source: 1
            shape: [256, 256]
            datatype: bool8
            byteorder: little

  -
    - An array where the data is stored in the first block in
      another ASDF file.
    - |
        !core/ndarray-1.0.0
          source: external.asdf
          shape: [256, 256]
          datatype: float64
          byteorder: little

definitions:
  scalar-datatype:
    description: |
      Describes the type of a single element.

      There is a set of numeric types, each with a single identifier:

      - `int8`, `int16`, `int32`, `int64`: Signed integer types, with
        the given bit size.

      - `uint8`, `uint16`, `uint32`, `uint64`: Unsigned integer types,
        with the given bit size.

      - `float32`: Single-precision floating-point type or "binary32",
        as defined in IEEE 754.

      - `float64`: Double-precision floating-point type or "binary64",
        as defined in IEEE 754.

      - `complex64`: Complex number where the real and imaginary parts
        are each single-precision floating-point ("binary32") numbers,
        as defined in IEEE 754.

```

```
    - `complex128`: Complex number where the real and imaginary
      parts are each double-precision floating-point ("binary64")
      numbers, as defined in IEEE 754.

    There are two distinct fixed-length string types, which must
    be indicated with a 2-element array where the first element is an
    identifier for the string type, and the second is a length:

    - `ascii`: A string containing ASCII text (all codepoints <
      128), where each character is 1 byte.

    - `ucs4`: A string containing unicode text in the UCS-4
      encoding, where each character is always 4 bytes long.  Here
      the number of bytes used is 4 times the given length.

  anyOf:
    - type: string
      enum: [int8, uint8, int16, uint16, int32, uint32, int64, uint64,
             float32, float64, complex64, complex128, bool8]
    - type: array
      items:
        - type: string
          enum: [ascii, ucs4]
        - type: integer
          minimum: 0
      minLength: 2
      maxLength: 2

datatype:
  description: |
    The data format of the array elements.  May be a single scalar
    datatype, or may be a nested list of datatypes.  When a list, each field
    may have a name.
  anyOf:
    - $ref: "#/definitions/scalar-datatype"
    - type: array
      items:
        anyOf:
          - $ref: "#/definitions/scalar-datatype"
          - type: object
            properties:
              name:
                type: string
                pattern: "[A-Za-z_][A-Za-z0-9_]*"
                description: The name of the field
              datatype:
                $ref: "#/definitions/datatype"
              byteorder:
                type: string
                enum: [big, little]
                description: |
                  The byteorder for the field.  If not provided, the
                  byteorder of the datatype as a whole will be used.
              shape:
                type: array
                items:
                  type: integer
                  minimum: 0
```

```
              required: [datatype]

  inline-data:
    description: |
      Inline data is stored in YAML format directly in the tree, rather than
      referencing a binary block.  It is made out of nested lists.

      If the datatype of the array is not specified, it is inferred from
      the array contents.  Type inference is supported only for
      homogeneous arrays, not tables.

      - If any of the elements in the array are YAML strings, the
        `datatype` of the entire array is `ucs4`, with the width of
        the largest string in the column, otherwise...

      - If any of the elements in the array are complex numbers, the
        `datatype` of the entire column is `complex128`, otherwise...

      - If any of the types in the column are numbers with a decimal
        point, the `datatype` of the entire column is `float64`,
        otherwise..

      - If any of the types in the column are integers, the `datatype`
        of the entire column is `int64`, otherwise...

      - The `datatype` of the entire column is `bool8`.

      Masked values may be included in the array using `null`.  If an
      explicit mask array is also provided, it takes precedence.

    type: array
    items:
      anyOf:
        - type: number
        - type: string
        - type: "null"
        - $ref: "complex-1.0.0"
        - $ref: "#/definitions/inline-data"
        - type: boolean

anyOf:
  - $ref: "#/definitions/inline-data"
  - type: object
    properties:
      source:
        description: |
          The source of the data.

          - If an integer: If positive, the zero-based index of the
            block within the same file. If negative, the index from
            the last block within the same file.  For example, a
            source of `-1` corresponds to the last block in the same
            file.

          - If a string, a URI to an external ASDF file containing the
            block data.  Relative URIs and ``file:`` and ``http:``
            protocols must be supported.  Other protocols may be supported
            by specific library implementations.
```

```
          The ability to reference block data in an external ASDF file
          is intentionally limited to the first block in the external
          ASDF file, and is intended only to support the needs of
          [exploded](ref:exploded).  For the more general case of
          referencing data in an external ASDF file, use tree
          [references](ref:references).

        anyOf:
          - type: integer
          - type: string
            format: uri

      data:
        description: |
          The data for the array inline.

          If `datatype` and/or `shape` are also provided, they must
          match the data here and can be used as a consistency check.
          `strides`, `offset` and `byteorder` are meaningless when
          `data` is provided.
        $ref: "#/definitions/inline-data"

      shape:
        description: |
          The shape of the array.

          The first entry may be the string `*`, indicating that the
          length of the first index of the array will be automatically
          determined from the size of the block.  This is used for
          streaming support.
        type: array
        items:
          anyOf:
            - type: integer
              minimum: 0
            - enum: ['*']

      datatype:
        description: |
          The data format of the array elements.
        $ref: "#/definitions/datatype"

      byteorder:
        description: >
          The byte order (big- or little-endian) of the array data.
        type: string
        enum: [big, little]

      offset:
        description: >
          The offset, in bytes, within the data for this start of this
          view.
        type: integer
        minimum: 0
        default: 0

      strides:
```

```yaml
    description: >
      The number of bytes to skip in each dimension.  If not provided,
      the array is assumed by be contiguous and in C order.  If
      provided, must be the same length as the shape property.
    type: array
    items:
      anyOf:
        - type: integer
          minimum: 1
        - type: integer
          maximum: -1

  mask:
    description: >
      Describes how missing values in the array are stored.  If a
      scalar number, that number is used to represent missing values.
      If an ndarray, the given array provides a mask, where non-zero
      values represent missing values in this array.  The mask array
      must be broadcastable to the dimensions of this array.
    anyOf:
      - type: number
      - $ref: "complex-1.0.0"
      - allOf:
        - $ref: "ndarray-1.0.0"
        - datatype: bool8

dependencies:
  source: [shape, datatype, byteorder]

propertyOrder: [source, data, mask, datatype, byteorder, shape, offset, strides]
```

5.1.4 table: A table.

Type: object.

A table.

A table is represented as a list of columns, where each entry is a *column* (page 37) object, containing the data and some additional information.

The data itself may be stored inline as text, or in binary in either row- or column-major order by use of the `strides` property on the individual column arrays.

Each column in the table must have the same first (slowest moving) dimension.

Properties:

`columns`

Type: array of (column-1.0.0 (page 37)).

A list of columns in the table.

Items:

Type: column-1.0.0 (page 37).

`meta`

Type: object.

Additional free-form metadata about the table.

Default: {}

Examples:

A table stored in column-major order, with each column in a separate block:

```
!core/table-1.0.0
  columns:
  - !core/column-1.0.0
    data: !core/ndarray-1.0.0
      source: 0
      datatype: float64
      byteorder: little
      shape: [3]
    description: RA
    meta: {foo: bar}
    name: a
    unit: !unit/unit-1.0.0 deg
  - !core/column-1.0.0
    data: !core/ndarray-1.0.0
      source: 1
      datatype: float64
      byteorder: little
      shape: [3]
    description: DEC
    name: b
  - !core/column-1.0.0
    data: !core/ndarray-1.0.0
      source: 2
      datatype: [ascii, 1]
      byteorder: big
      shape: [3]
    description: The target name
    name: c
```

A table stored in row-major order, all stored in the same block:

```
!core/table-1.0.0
  columns:
  - !core/column-1.0.0
    data: !core/ndarray-1.0.0
      source: 0
      datatype: float64
      byteorder: little
      shape: [3]
      strides: [13]
    description: RA
    meta: {foo: bar}
    name: a
    unit: !unit/unit-1.0.0 deg
  - !core/column-1.0.0
    data: !core/ndarray-1.0.0
      source: 0
      datatype: float64
      byteorder: little
      shape: [3]
      offset: 4
      strides: [13]
```

```
    description: DEC
    name: b
  - !core/column-1.0.0
    data: !core/ndarray-1.0.0
      source: 0
      datatype: [ascii, 1]
      byteorder: big
      shape: [3]
      offset: 12
      strides: [13]
    description: The target name
    name: c
```

Original schema in YAML

```
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/core/table-1.0.0"
tag: "tag:stsci.edu:asdf/core/table-1.0.0"

title: >
  A table.

description: |
  A table is represented as a list of columns, where each entry is a
  [column](ref:http://stsci.edu/schemas/asdf/core/column-1.0.0)
  object, containing the data and some additional information.

  The data itself may be stored inline as text, or in binary in either
  row- or column-major order by use of the `strides` property on the
  individual column arrays.

  Each column in the table must have the same first (slowest moving)
  dimension.

examples:
  -
    - A table stored in column-major order, with each column in a separate block
    - |
        !core/table-1.0.0
          columns:
          - !core/column-1.0.0
            data: !core/ndarray-1.0.0
              source: 0
              datatype: float64
              byteorder: little
              shape: [3]
            description: RA
            meta: {foo: bar}
            name: a
            unit: !unit/unit-1.0.0 deg
          - !core/column-1.0.0
            data: !core/ndarray-1.0.0
              source: 1
              datatype: float64
```

```
            byteorder: little
            shape: [3]
          description: DEC
          name: b
        - !core/column-1.0.0
          data: !core/ndarray-1.0.0
            source: 2
            datatype: [ascii, 1]
            byteorder: big
            shape: [3]
          description: The target name
          name: c

  -
    - A table stored in row-major order, all stored in the same block
    - |
        !core/table-1.0.0
          columns:
          - !core/column-1.0.0
            data: !core/ndarray-1.0.0
              source: 0
              datatype: float64
              byteorder: little
              shape: [3]
              strides: [13]
            description: RA
            meta: {foo: bar}
            name: a
            unit: !unit/unit-1.0.0 deg
          - !core/column-1.0.0
            data: !core/ndarray-1.0.0
              source: 0
              datatype: float64
              byteorder: little
              shape: [3]
              offset: 4
              strides: [13]
            description: DEC
            name: b
          - !core/column-1.0.0
            data: !core/ndarray-1.0.0
              source: 0
              datatype: [ascii, 1]
              byteorder: big
              shape: [3]
              offset: 12
              strides: [13]
            description: The target name
            name: c

type: object
properties:
  columns:
    description: |
      A list of columns in the table.
    type: array
    items:
      $ref: column-1.0.0
```

```
  meta:
    description: |
      Additional free-form metadata about the table.
    type: object
    default: {}

additionalProperties: false
requiredProperties: [data]
```

5.1.5 column: A column in a table.

Type: object.

A column in a table.

Each column contains a name and an array of data, and an optional description and unit.

Properties:

`name`

Type: string (regex `[A-Za-z_][A-Za-z0-9_]*`).

The name of the column. Each name in a table (http://stsci.edu/schemas/asdf/core/table-1.0.0) must be unique.

`data`

Type: ndarray-1.0.0 (page 20).

The array data for the column.

`description`

Type: string.

An optional description of the column.

Default: ""

`unit`

Type: unit-1.0.0 (page 46).

An optional unit for the column.

`meta`

Type: object.

Additional free-form metadata about the column.

Default: {}

Original schema in YAML

```
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/core/column-1.0.0"
tag: "tag:stsci.edu:asdf/core/column-1.0.0"
```

```

title: >
  A column in a table.

description: |
  Each column contains a name and an array of data, and an optional description
  and unit.

type: object
properties:
  name:
    description: |
      The name of the column.  Each name in a
      [table](http://stsci.edu/schemas/asdf/core/table-1.0.0) must be
      unique.
    type: string
    pattern: "[A-Za-z_][A-Za-z0-9_]*"

  data:
    description: |
      The array data for the column.
    allOf:
      - $ref: ndarray-1.0.0

  description:
    description: |
      An optional description of the column.
    type: string
    default: ''

  unit:
    description:
      An optional unit for the column.
    allOf:
      - $ref: ../unit/unit-1.0.0

  meta:
    description:
      Additional free-form metadata about the column.
    type: object
    default: {}

requiredProperties: [name, data]
additionalProperties: false
```

5.1.6 constant: Specify that a value is a constant.

Type: any.

Specify that a value is a constant.

Used as a utility to indicate that value is a literal constant.

Original schema in YAML

```
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/core/constant-1.0.0"
tag: "tag:stsci.edu:asdf/core/constant-1.0.0"
title: Specify that a value is a constant.
description: |
  Used as a utility to indicate that value is a literal constant.
```

5.1.7 software: Describes a software package.

Type: object.

Describes a software package.

Properties:

`name`

Type: string. Required.

The name of the application or library.

`author`

Type: string. Required.

The author (or institution) that produced the software package.

`homepage`

Type: string (format uri). Required.

A URI to the homepage of the software.

`version`

Type: string. Required.

The version of the software used. It is recommended, but not required, that this follows the (Semantic Versioning Specification)[http://semver.org/spec/v2.0.0.html].

Original schema in YAML

```
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/core/software-1.0.0"
title: |
  Describes a software package.

tag: "tag:stsci.edu:asdf/core/software-1.0.0"
type: object
properties:
  name:
    description: |
      The name of the application or library.
    type: string
```

```
  author:
    description: |
      The author (or institution) that produced the software package.
    type: string

  homepage:
    description: |
      A URI to the homepage of the software.
    type: string
    format: uri

  version:
    description: |
      The version of the software used.  It is recommended, but not
      required, that this follows the (Semantic Versioning
      Specification)[http://semver.org/spec/v2.0.0.html].
    type: string

required: [name, author, homepage, version]
additionalProperties: true
```

5.1.8 history_entry: An entry in the file history.

Type: object.

An entry in the file history.

Properties:

`description`

Type: string.

A description of the transformation performed.

`time`

Type: string (format date-time).

A timestamp for the operation, in UTC.

`software`

Type: software-1.0.0 (page 39) or array of (software-1.0.0 (page 39)).

One or more descriptions of the software that performed the operation.

Any of:

Type: software-1.0.0 (page 39).

Type: array of (software-1.0.0 (page 39)).

Items:

Type: software-1.0.0 (page 39).

Original schema in YAML

```yaml
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/core/history_entry-1.0.0"
title: |
  An entry in the file history.

tag: "tag:stsci.edu:asdf/core/history_entry-1.0.0"
type: object
properties:
  description:
    description: |
      A description of the transformation performed.
    type: string

  time:
    description: |
      A timestamp for the operation, in UTC.
    type: string
    format: date-time

  software:
    description: |
      One or more descriptions of the software that performed the
      operation.
    anyOf:
      - $ref: "software-1.0.0"
      - type: array
        items:
          $ref: "software-1.0.0"

requiredProperties: [description]
additionalProperties: true
```

5.2 FITS

The `fits` module contains schema that support backward compatibility with FITS.

Requires:

Core (page 17)

5.2.1 fits: A FITS file inside of an ASDF file.

Type: array of (object).

A FITS file inside of an ASDF file.

This schema is useful for distributing ASDF files that can automatically be converted to FITS files by specifying the exact content of the resulting FITS file.

Not all kinds of data in FITS are directly representable in ASDF. For example, applying an offset and scale to the data using the `BZERO` and `BSCALE` keywords. In these cases, it will not be possible to store the data in the native format from FITS and also be accessible in its proper form in the ASDF file.

Only image and binary table extensions are supported.

Items:

Type: object.

Each item represents a single header/data unit (HDU).

Properties:

`header`

Type: array of (array $0 \leq len \leq 3$). Required.

A list of the keyword/value/comment triples from the header, in the order they appear in the FITS file.

Items:

Type: array $0 \leq len \leq 3$.

Items:

`index[0]`

Type: string ($len \leq 8$ regex `[A-Z0-9]*`).

The keyword.

`index[1]`

Type: string ($len \leq 60$) or number or boolean.

The value.

Any of:

Type: string ($len \leq 60$).

Type: number.

Type: boolean.

`index[2]`

Type: string ($len \leq 60$).

The comment.

`data`

Type: ndarray-1.0.0 (page 20) or table-1.0.0 (page 33) or null.

The data part of the HDU.

Default: null

Any of:

Type: ndarray-1.0.0 (page 20).

Type: table-1.0.0 (page 33).

Type: null.

Examples:

A simple FITS file with a primary header and two extensions:

```
!fits/fits-1.0.0
    - header:
      - [SIMPLE, true, conforms to FITS standard]
      - [BITPIX, 8, array data type]
      - [NAXIS, 0, number of array dimensions]
      - [EXTEND, true]
      - []
      - ['', Top Level MIRI Metadata]
      - []
      - [DATE, '2013-08-30T10:49:55.070373', The date this file was created (UTC)]
      - [FILENAME, MiriDarkReferenceModel_test.fits, The name of the file]
      - [TELESCOP, JWST, The telescope used to acquire the data]
      - []
      - ['', Information about the observation]
      - []
      - [DATE-OBS, '2013-08-30T10:49:55.000000', The date the observation was made (UTC)]
    - data: !core/ndarray-1.0.0
        datatype: float32
        shape: [2, 3, 3, 4]
        source: 0
        byteorder: big
      header:
      - [XTENSION, IMAGE, Image extension]
      - [BITPIX, -32, array data type]
      - [NAXIS, 4, number of array dimensions]
      - [NAXIS1, 4]
      - [NAXIS2, 3]
      - [NAXIS3, 3]
      - [NAXIS4, 2]
      - [PCOUNT, 0, number of parameters]
      - [GCOUNT, 1, number of groups]
      - [EXTNAME, SCI, extension name]
      - [BUNIT, DN, Units of the data array]
    - data: !core/ndarray-1.0.0
        datatype: float32
        shape: [2, 3, 3, 4]
        source: 1
        byteorder: big
      header:
      - [XTENSION, IMAGE, Image extension]
      - [BITPIX, -32, array data type]
      - [NAXIS, 4, number of array dimensions]
      - [NAXIS1, 4]
      - [NAXIS2, 3]
      - [NAXIS3, 3]
      - [NAXIS4, 2]
      - [PCOUNT, 0, number of parameters]
      - [GCOUNT, 1, number of groups]
      - [EXTNAME, ERR, extension name]
      - [BUNIT, DN, Units of the error array]
```

Original schema in YAML

```
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/fits/fits-1.0.0"
title: >
  A FITS file inside of an ASDF file.
description: |
  This schema is useful for distributing ASDF files that can
  automatically be converted to FITS files by specifying the exact
  content of the resulting FITS file.

  Not all kinds of data in FITS are directly representable in ASDF.
  For example, applying an offset and scale to the data using the
  `BZERO` and `BSCALE` keywords.  In these cases, it will not be
  possible to store the data in the native format from FITS and also
  be accessible in its proper form in the ASDF file.

  Only image and binary table extensions are supported.

examples:
  -
    - A simple FITS file with a primary header and two extensions
    - |
        !fits/fits-1.0.0
            - header:
              - [SIMPLE, true, conforms to FITS standard]
              - [BITPIX, 8, array data type]
              - [NAXIS, 0, number of array dimensions]
              - [EXTEND, true]
              - []
              - ['', Top Level MIRI Metadata]
              - []
              - [DATE, '2013-08-30T10:49:55.070373', The date this file was created (UTC)]
              - [FILENAME, MiriDarkReferenceModel_test.fits, The name of the file]
              - [TELESCOP, JWST, The telescope used to acquire the data]
              - []
              - ['', Information about the observation]
              - []
              - [DATE-OBS, '2013-08-30T10:49:55.000000', The date the observation was made (UTC)]
            - data: !core/ndarray-1.0.0
                datatype: float32
                shape: [2, 3, 3, 4]
                source: 0
                byteorder: big
              header:
              - [XTENSION, IMAGE, Image extension]
              - [BITPIX, -32, array data type]
              - [NAXIS, 4, number of array dimensions]
              - [NAXIS1, 4]
              - [NAXIS2, 3]
              - [NAXIS3, 3]
              - [NAXIS4, 2]
              - [PCOUNT, 0, number of parameters]
              - [GCOUNT, 1, number of groups]
              - [EXTNAME, SCI, extension name]
              - [BUNIT, DN, Units of the data array]
```

```
          - data: !core/ndarray-1.0.0
              datatype: float32
              shape: [2, 3, 3, 4]
              source: 1
              byteorder: big
            header:
            - [XTENSION, IMAGE, Image extension]
            - [BITPIX, -32, array data type]
            - [NAXIS, 4, number of array dimensions]
            - [NAXIS1, 4]
            - [NAXIS2, 3]
            - [NAXIS3, 3]
            - [NAXIS4, 2]
            - [PCOUNT, 0, number of parameters]
            - [GCOUNT, 1, number of groups]
            - [EXTNAME, ERR, extension name]
            - [BUNIT, DN, Units of the error array]

tag: "tag:stsci.edu:asdf/fits/fits-1.0.0"
type: array
items:
  description: >
    Each item represents a single header/data unit (HDU).
  type: object
  properties:
    header:
      description: >
        A list of the keyword/value/comment triples from the header,
        in the order they appear in the FITS file.
      type: array
      items:
        type: array
        minItems: 0
        maxItems: 3
        items:
          - description: "The keyword."
            type: string
            maxLength: 8
            pattern: "[A-Z0-9]*"
          - description: "The value."
            anyOf:
              - type: string
                maxLength: 60
              - type: number
              - type: boolean
          - description: "The comment."
            type: string
            maxLength: 60
    data:
      description: "The data part of the HDU."
      anyOf:
        - $ref: "../core/ndarray-1.0.0"
        - $ref: "../core/table-1.0.0"
        - type: "null"
      default: null
  required: [header]
  additionalProperties: false
```

5.3 Unit

The `unit` module contains schema to support the units of physical quantities.

Requires:

Core (page 17)

5.3.1 unit: Physical unit.

Type: any or any.

Physical unit.

This represents a physical unit, in VOUnit syntax, Version 1.0 (http://www.ivoa.net/documents/VOUnits/index.html). Where units are not explicitly tagged, they are assumed to be in VOUnit syntax.

Any of:

Type: any.

Type: any.

Examples:

Example unit:

```
!unit/unit-1.0.0 "2.1798721 10-18kg m2 s-2"
```

Original schema in YAML

```yaml
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/unit/unit-1.0.0"
title: Physical unit.
description: >
  This represents a physical unit, in [VOUnit syntax, Version
  1.0](http://www.ivoa.net/documents/VOUnits/index.html).

  Where units are not explicitly tagged, they are assumed to be
  in VOUnit syntax.
examples:
  -
    - Example unit
    - |
      !unit/unit-1.0.0 "2.1798721 10-18kg m2 s-2"

anyOf:
  - tag: "tag:stsci.edu:asdf/unit/unit-1.0.0"
  - {}

type: string
pattern: "[\x00-\x7f]*"
```

5.3.2 defunit: Define a new physical unit.

Type: object.

Define a new physical unit.

Defines a new unit. It can be used to either:

- Define a new base unit.
- Create a new unit name that is a equivalent to a given unit.

The new unit must be defined before any unit tags that use it.

Properties:

`name`

Type: string (regex `[A-Za-z_][A-Za-z0-9_]+`). Required.

The name of the new unit.

`unit`

Type: unit-1.0.0 (page 46) or null.

The unit that the new name is equivalent to. It is optional, and if not provided, or `null`, this `defunit` defines a new base unit.

Any of:

Type: unit-1.0.0 (page 46).

Type: null.

Original schema in YAML


```yaml
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/unit/defunit-1.0.0"
title: Define a new physical unit.
description: |
  Defines a new unit.  It can be used to either:

  - Define a new base unit.

  - Create a new unit name that is a equivalent to a given unit.

  The new unit must be defined before any unit tags that use it.

tag: "tag:stsci.edu:asdf/unit/defunit-1.0.0"
type: object
properties:
  name:
    description: The name of the new unit.
    type: string
    pattern: "[A-Za-z_][A-Za-z0-9_]+"

```

```
  unit:
    description: |
      The unit that the new name is equivalent to.  It is optional,
      and if not provided, or ``null``, this ``defunit`` defines a new
      base unit.

    anyOf:
      - $ref: "unit-1.0.0"
      - type: "null"

required: [name]
```

5.4 Time

The time module contains schema to support representing instances in time and time deltas.

Requires:

Core (page 17)

5.4.1 time: Represents an instance in time.

Type: any and *definitions/string_formats* (page 49) or *definitions/array_of_strings* (page 49) or object or object.

Represents an instance in time.

A "time" is a single instant in time. It may explicitly specify the way time is represented (the "format") and the "scale" which specifies the offset and scaling relation of the unit of time.

Specific emphasis is placed on supporting time scales (e.g. UTC, TAI, UT1, TDB) and time representations (e.g. JD, MJD, ISO 8601) that are used in astronomy and required to calculate, e.g., sidereal times and barycentric corrections.

Times may be represented as one of the following:

- an object, with explicit `value`, and optional `format`, `scale` and `location`.
- a string, in which case the format is guessed from across the unambiguous options (`iso`, `byear`, `jyear`, `yday`), and the scale is hardcoded to `UTC`.

In either case, a single time tag may be used to represent an n-dimensional array of times, using either an `ndarray` tag or inline as (possibly nested) YAML lists. If YAML lists, the same format must be used for all time values.

The precision of the numeric formats should only be assumed to be as good as an IEEE-754 double precision (float64) value. If higher-precision is required, the `iso` or `yday` format should be used.

Definitions:

`iso_time`

Type: string (regex `[0-9]{4}-(0[1-9])|(1[0-2])-(0[1-9])|([1-2][0-9])|(3[0-1])[T ]([0-1][0-9])|(2[0-4]):[0-5][0-9]:[0-5][0-9](.[0-9]+)?`).

`byear`

Type: string (regex `B[0-9]+(.[0-9]+)?`).

`jyear`

Type: string (regex `J[0-9]+(.[0-9]+)?`).

`yday`

Type: string (regex `[0-9]{4}:(00[1-9])|(0[1-9][0-9])|([1-2][0-9][0-9])|(3[0-5][0-9])|(36[0-5]):([0-1][0-9])|(`
).

`string_formats`

Type: *definitions/iso_time* (page 48) or *definitions/byear* (page 48) or *definitions/jyear* (page 48) or *definitions/yday* (page 48).

Any of:

Type: *definitions/iso_time* (page 48).

Type: *definitions/byear* (page 48).

Type: *definitions/jyear* (page 48).

Type: *definitions/yday* (page 48).

`array_of_strings`

Type: array of (*definitions/array_of_strings* (page 49) or *definitions/string_formats* (page 49)).

Items:

Type: *definitions/array_of_strings* (page 49) or *definitions/string_formats* (page 49).

Any of:

Type: *definitions/array_of_strings* (page 49).

Type: *definitions/string_formats* (page 49).

All of:

`0`

Type: any.

`1`

Type: *definitions/string_formats* (page 49) or *definitions/array_of_strings* (page 49) or object or object.

Any of:

Type: *definitions/string_formats* (page 49).

Type: *definitions/array_of_strings* (page 49).

Type: object.

Properties:

Type: object.

Properties:

`value`

Type: *definitions/string_formats* (page 49) or *definitions/array_of_strings* (page 49) or ndarray-1.0.0 (page 20) or number. Required.

The value(s) of the time.

Any of:

Type: *definitions/string_formats* (page 49).

Type: *definitions/array_of_strings* (page 49).

Type: ndarray-1.0.0 (page 20).

Type: number.

`format`

Type: any from ["iso", "yday", "byear", "jyear", "decimalyear", "jd", "mjd", "gps", "unix", "cxcsec"].

The format of the time.

If not provided, the the format should be guessed from the string from among the following unambiguous options: `iso`, `byear`, `jyear` and `yday`.

The supported formats are:

- `iso`: ISO 8601 compliant date-time format `YYYY-MM- DDTHH:MM:SS.sss...`. For example, `2000-01-01 00:00:00.000` is midnight on January 1, #. The `T` separating the date from the time section is optional.
- `yday`: Year, day-of-year and time as `YYYY:DOY:HH:MM:SS.sss...`. The day-of-year (DOY) goes from 001 to 365 (366 in leap years). For example, `2000:001:00:00:00.000` is midnight on January 1, 2000.
- `byear`: Besselian Epoch year, eg. `B1950.0`. The `B` is optional if the `byear` format is explicitly specified.
- `jyear`: Julian Epoch year, eg. `J2000.0`. The `J` is optional if the `jyear` format is explicitly specified.
- `decimalyear`: Time as a decimal year, with integer values corresponding to midnight of the first day of each year. For example 2000.5 corresponds to the ISO time `2000-07-02 00:00:00`.
- `jd`: Julian Date time format. This represents the number of days since the beginning of the Julian Period. For example, 2451544.5 in `jd` is midnight on January 1, 2000.
- `mjd`: Modified Julian Date time format. This represents the number of days since midnight on November 17, 1858. For example, 51544.0 in MJD is midnight on January 1, 2000.
- `gps`: GPS time: seconds from 1980-01-06 00:00:00 UTC For example, 630720013.0 is midnight on January 1, 2000.

- unix: Unix time: seconds from 1970-01-01 00:00:00 UTC. For example, 946684800.0 in Unix time is midnight on January 1, 2000. [TODO: Astropy's definition of UNIX time doesn't match POSIX's here. What should we do for the purposes of ASDF?]

scale

Type: any from ["utc", "tai", "tcb", "tcg", "tdb", "tt", "ut1"].

The time scale (or time standard) is a specification for measuring time: either the rate at which time passes; or points in time; or both. See also [3] and [4].

These scales are defined in detail in SOFA Time Scale and Calendar Tools (http://www.iausofa.org/sofa_ts_c.pdf).

The supported time scales are:

- utc: Coordinated Universal Time (UTC). This is the default time scale, except for gps, unix.
- tai: International Atomic Time (TAI).
- tcb: Barycentric Coordinate Time (TCB).
- tcg: Geocentric Coordinate Time (TCG).
- tdb: Barycentric Dynamical Time (TDB).
- tt: Terrestrial Time (TT).
- ut1: Universal Time (UT1).

location

Type: object or object.

Specifies the observer location for scales that are sensitive to observer location, currently only tdb. May be specified either with geocentric coordinates (X, Y, Z) with an optional unit or geodetic coordinates:

- long: longitude in degrees
- lat: in degrees
- h: optional height

Any of:

Type: object.

Properties:

x

Type: number. Required.

y

Type: number. Required.

z

Type: number. Required.

unit

Type: unit-1.0.0 (page 46) and any.

All of:

`0`

Type: unit-1.0.0 (page 46).

`1`

Type: any.

Default: “m”

`---`

Type: object.

Properties:

`long`

Type: number -180 $\leq x \leq$ 180. Required.

`lat`

Type: number -90 $\leq x \leq$ 90. Required.

`h`

Type: number.

Default: 0

`unit`

Type: unit-1.0.0 (page 46) and any.

All of:

`0`

Type: unit-1.0.0 (page 46).

`1`

Type: any.

Default: “m”

Examples:

Example ISO time:

```
!time/time-1.0.0 "2000-12-31T13:05:27.737"
```

Example year, day-of-year and time format time:

```
!time/time-1.0.0 "2001:003:04:05:06.789"
```

Example Besselian Epoch time:

```
!time/time-1.0.0 B2000.0
```

Example Besselian Epoch time, equivalent to above:

```
!time/time-1.0.0
  value: 2000.0
  format: byear
```

Example list of times:

```
!time/time-1.0.0
  ["2000-12-31T13:05:27.737", "2000-12-31T13:06:38.444"]
```

Example of an array of times:

```
!time/time-1.0.0
  value: !core/ndarray-1.0.0
    data: [2000, 2001]
    datatype: float64
  format: jyear
```

Example with a location:

```
!time/time-1.0.0
  value: 2000.0
  format: jyear
  scale: tdb
  location:
    x: 6378100
    y: 0
    z: 0
```

Original schema in YAML

```
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/asdf/asdf-schema-1.0.0"
id: "http://stsci.edu/schemas/asdf/time/time-1.0.0"
title: Represents an instance in time.
description: |
  A "time" is a single instant in time.  It may explicitly specify the
  way time is represented (the "format") and the "scale" which
  specifies the offset and scaling relation of the unit of time.

  Specific emphasis is placed on supporting time scales (e.g. UTC,
  TAI, UT1, TDB) and time representations (e.g. JD, MJD, ISO 8601)
  that are used in astronomy and required to calculate, e.g., sidereal
  times and barycentric corrections.

  Times may be represented as one of the following:

  - an object, with explicit `value`, and optional `format`, `scale`
    and `location`.

  - a string, in which case the format is guessed from across
    the unambiguous options (`iso`, `byear`, `jyear`, `yday`), and the
    scale is hardcoded to `UTC`.

  In either case, a single time tag may be used to represent an
  n-dimensional array of times, using either an `ndarray` tag or
  inline as (possibly nested) YAML lists.  If YAML lists, the same
  format must be used for all time values.

  The precision of the numeric formats should only be assumed to be as
  good as an IEEE-754 double precision (float64) value.  If
  higher-precision is required, the `iso` or `yday` format should be
```

```
  used.

examples:
  -
    - Example ISO time
    - |
        !time/time-1.0.0 "2000-12-31T13:05:27.737"

  -
    - Example year, day-of-year and time format time
    - |
        !time/time-1.0.0 "2001:003:04:05:06.789"

  -
    - Example Besselian Epoch time
    - |
        !time/time-1.0.0 B2000.0

  -
    - Example Besselian Epoch time, equivalent to above
    - |
        !time/time-1.0.0
          value: 2000.0
          format: byear

  -
    - Example list of times
    - |
        !time/time-1.0.0
          ["2000-12-31T13:05:27.737", "2000-12-31T13:06:38.444"]

  -
    - Example of an array of times
    - |
        !time/time-1.0.0
          value: !core/ndarray-1.0.0
            data: [2000, 2001]
            datatype: float64
          format: jyear

  -
    - Example with a location
    - |
        !time/time-1.0.0
          value: 2000.0
          format: jyear
          scale: tdb
          location:
            x: 6378100
            y: 0
            z: 0

definitions:
  iso_time:
    type: string
    pattern: "[0-9]{4}-(0[1-9])|(1[0-2])-(0[1-9])|([1-2][0-9])|(3[0-1])[T ]([0-1][0-9])|(2[0-4]):[0-5][0-9]:[0-5][0-9](.[

  byear:
```

```
    type: string
    pattern: "B[0-9]+(.[0-9]+)?"

  jyear:
    type: string
    pattern: "J[0-9]+(.[0-9]+)?"

  yday:
    type: string
    pattern: "[0-9]{4}:(00[1-9])|(0[1-9][0-9])|([1-2][0-9][0-9])|(3[0-5][0-9])|(36[0-5]):([0-1][0-9])|([0-1][0-9])|(2[0-4

  string_formats:
    anyOf:
      - $ref: "#/definitions/iso_time"
      - $ref: "#/definitions/byear"
      - $ref: "#/definitions/jyear"
      - $ref: "#/definitions/yday"

  array_of_strings:
    type: array
    items:
      anyOf:
        - $ref: "#/definitions/array_of_strings"
        - $ref: "#/definitions/string_formats"

allOf:
  - tag: "tag:stsci.edu:asdf/time/time-1.0.0"
  - anyOf:
    - $ref: "#/definitions/string_formats"

    - $ref: "#/definitions/array_of_strings"

    - type: object
      properties:
        $ref: "../core/ndarray-1.0.0#anyOf/1/properties"

    - type: object
      properties:
        value:
          description: |
            The value(s) of the time.

          anyOf:
            - $ref: "#/definitions/string_formats"
            - $ref: "#/definitions/array_of_strings"
            - $ref: "../core/ndarray-1.0.0"
            - type: number

        format:
          description: |
            The format of the time.

            If not provided, the the format should be guessed from the
            string from among the following unambiguous options:
            `iso`, `byear`, `jyear` and `yday`.

            The supported formats are:

```

```
            - `iso`: ISO 8601 compliant date-time format
              `YYYY-MM-DDTHH:MM:SS.sss...`.  For example,
              `2000-01-01 00:00:00.000` is midnight on January 1,
              2000.  The `T` separating the date from the time
              section is optional.

            - `yday`: Year, day-of-year and time as
              `YYYY:DOY:HH:MM:SS.sss...`. The day-of-year (DOY) goes
              from 001 to 365 (366 in leap years). For example,
              `2000:001:00:00:00.000` is midnight on January 1,
              2000.

            - `byear`: Besselian Epoch year, eg. `B1950.0`.  The `B`
              is optional if the `byear` format is explicitly
              specified.

            - `jyear`: Julian Epoch year, eg. `J2000.0`.  The `J` is
              optional if the `jyear` format is explicitly
              specified.

            - `decimalyear`: Time as a decimal year, with integer
              values corresponding to midnight of the first day of
              each year. For example 2000.5 corresponds to the ISO
              time `2000-07-02 00:00:00`.

            - `jd`: Julian Date time format. This represents the
              number of days since the beginning of the Julian
              Period. For example, 2451544.5 in `jd` is midnight on
              January 1, 2000.

            - `mjd`: Modified Julian Date time format. This
              represents the number of days since midnight on
              November 17, 1858. For example, 51544.0 in MJD is
              midnight on January 1, 2000.

            - `gps`: GPS time: seconds from 1980-01-06 00:00:00 UTC
              For example, 630720013.0 is midnight on January 1,
              2000.

            - `unix`: Unix time: seconds from 1970-01-01 00:00:00
              UTC. For example, 946684800.0 in Unix time is midnight
              on January 1, 2000.  [TODO: Astropy's definition of
              UNIX time doesn't match POSIX's here.  What should we
              do for the purposes of ASDF?]

          enum:
            - iso
            - yday
            - byear
            - jyear
            - decimalyear
            - jd
            - mjd
            - gps
            - unix
            - cxcsec

        scale:
```

```
      description: |
        The time scale (or time standard) is a specification for
        measuring time: either the rate at which time passes; or
        points in time; or both. See also [3] and [4].

        These scales are defined in detail in [SOFA Time Scale and
        Calendar Tools](http://www.iausofa.org/sofa_ts_c.pdf).

        The supported time scales are:

        - `utc`: Coordinated Universal Time (UTC).  This is the
          default time scale, except for `gps`, `unix`.

        - `tai`: International Atomic Time (TAI).

        - `tcb`: Barycentric Coordinate Time (TCB).

        - `tcg`: Geocentric Coordinate Time (TCG).

        - `tdb`: Barycentric Dynamical Time (TDB).

        - `tt`: Terrestrial Time (TT).

        - `ut1`: Universal Time (UT1).

      enum:
        - utc
        - tai
        - tcb
        - tcg
        - tdb
        - tt
        - ut1

    location:
      description: |
        Specifies the observer location for scales that are
        sensitive to observer location, currently only `tdb`.  May
        be specified either with geocentric coordinates (X, Y, Z)
        with an optional unit or geodetic coordinates:
          - `long`: longitude in degrees
          - `lat`: in degrees
          - `h`: optional height

      anyOf:
        - type: object
          properties:
            x:
              type: number
            y:
              type: number
            z:
              type: number
            unit:
              allOf:
                - $ref: "../unit/unit-1.0.0"
                - default: m
          required: [x, y, z]
```

```
          - type: object
            properties:
              long:
                type: number
                minimum: -180
                maximum: 180
              lat:
                type: number
                minimum: -90
                maximum: 90
              h:
                type: number
                default: 0
              unit:
                allOf:
                  - $ref: "../unit/unit-1.0.0"
                  - default: m
            required: [long, lat]

    required: [value]
```

5.5 Transform

The `transform` module contains schema used to describe transformations.

Requires:

Core (page 17)

5.5.1 Basics

transform: A generic type used to mark where other transforms are accepted.

Type: object.

A generic type used to mark where other transforms are accepted.

These objects are designed to be nested in arbitrary ways to build up transformation pipelines out of a number of low-level pieces.

Properties:

`name`

Type: string.

A user-friendly name for the transform, to give it extra meaning.

`domain`

Type: array of (domain-1.0.0 (page 62)).

The domain (range of valid inputs) to the transform. Each entry in the list corresponds to an input dimension.

Items:

Type: domain-1.0.0 (page 62).

inverse

Type: transform-1.0.0 (page 58).

Explicitly sets the inverse transform of this transform.

If the transform has a direct analytic inverse, this property is usually not necessary, as the ASDF-reading tool can provide it automatically.

Original schema in YAML

```yaml
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/transform/transform-1.0.0"
title: >
  A generic type used to mark where other transforms are accepted.

description: >
  These objects are designed to be nested in arbitrary ways to build up
  transformation pipelines out of a number of low-level pieces.

type: object
properties:
  name:
    description: |
      A user-friendly name for the transform, to give it extra
      meaning.
    type: string

  domain:
    description: |
      The domain (range of valid inputs) to the transform.
      Each entry in the list corresponds to an input dimension.
    type: array
    items:
      $ref: "domain-1.0.0"

  inverse:
    description: |
      Explicitly sets the inverse transform of this transform.

      If the transform has a direct analytic inverse, this
      property is usually not necessary, as the ASDF-reading tool
      can provide it automatically.

    $ref: "transform-1.0.0"
additionalProperties: true
```

generic: A generic transform.

Type: transform-1.0.0 (page 58) and object.

A generic transform.

This is used **entirely** for bootstrapping purposes so one can create composite models including transforms that haven't yet been written. **IT WILL NOT BE IN THE FINAL VERSION OF THE SPEC.**

All of:

0

Type: transform-1.0.0 (page 58).

1

Type: object.

Properties:

n_inputs

Type: integer. Required.

n_outputs

Type: integer. Required.

Original schema in YAML

```
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/transform/generic-1.0.0"
tag: "tag:stsci.edu:asdf/transform/generic-1.0.0"
title: >
  A generic transform.
description: >
  This is used **entirely** for bootstrapping purposes so one can
  create composite models including transforms that haven't yet been
  written.  **IT WILL NOT BE IN THE FINAL VERSION OF THE SPEC**.

allOf:
  - $ref: "transform-1.0.0"
  - type: object
    properties:
      n_inputs:
        type: integer
      n_outputs:
        type: integer
    required: [n_inputs, n_outputs]
```

identity: The identity transform.

Type: transform-1.0.0 (page 58) and object.

The identity transform.

Invertibility: The inverse of this transform is also the identity transform.

All of:

0

Type: transform-1.0.0 (page 58).

1

Type: object.

Properties:

`n_dims`

Type: integer.

The number of dimensions.

Default: 1

Original schema in YAML

```
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/transform/identity-1.0.0"
tag: "tag:stsci.edu:asdf/transform/identity-1.0.0"
title: >
  The identity transform.
description: >
  Invertibility: The inverse of this transform is also the identity
  transform.
allOf:
  - $ref: "transform-1.0.0"
  - type: object
    properties:
      n_dims:
        type: integer
        default: 1
        description: |
          The number of dimensions.
```

constant: A transform that takes no inputs and always outputs a constant value.

Type: transform-1.0.0 (page 58) and object.

A transform that takes no inputs and always outputs a constant value.

Invertibility: All ASDF tools are required to be able to compute the analytic inverse of this transform, which always outputs zero values.

All of:

`0`

Type: transform-1.0.0 (page 58).

`1`

Type: object.

Properties:

`value`

Type: number. Required.

Original schema in YAML

```
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/transform/constant-1.0.0"
tag: "tag:stsci.edu:asdf/transform/constant-1.0.0"
title: >
  A transform that takes no inputs and always outputs a constant
  value.
description: |
  Invertibility: All ASDF tools are required to be able to compute the
  analytic inverse of this transform, which always outputs zero values.
allOf:
  - $ref: "transform-1.0.0"
  - type: object
    properties:
      value:
        type: number
    required: [value]
```

domain: Defines the domain of an input axis.

Type: any.

Defines the domain of an input axis.

Describes the range of acceptable input values to a particular axis of a transform.

Examples:

The domain `[0, 1)`.:

```
!transform/domain-1.0.0
  lower: 0
  upper: 1
  includes_lower: true
```

Original schema in YAML

```
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/transform/domain-1.0.0"
tag: "tag:stsci.edu:asdf/transform/domain-1.0.0"
title: >
  Defines the domain of an input axis.

description: >
  Describes the range of acceptable input values to a particular
  axis of a transform.

examples:
  -
    - The domain `[0, 1)`.
    - |
```

```
    !transform/domain-1.0.0
      lower: 0
      upper: 1
      includes_lower: true

properties:
  lower:
    description: >
      The lower value of the domain.  If not provided, the
      domain has no lower limit.
    type: number
    default: -.inf

  upper:
    description: >
      The upper value of the domain.  If not provided, the
      domain has no upper limit.
    type: number
    default: .inf

  includes_lower:
    description: If `true`, the domain includes `lower`.
    type: boolean
    default: false

  includes_upper:
    description: If `true`, the domain includes `upper`.
    type: boolean
    default: false
```

5.5.2 Compound transformations

compose: Perform a list of subtransforms in series.

Type: transform-1.0.0 (page 58) and any.

Perform a list of subtransforms in series.

The output of each subtransform is fed into the input of the next subtransform.

The number of output dimensions of each subtransform must be equal to the number of input dimensions of the next subtransform in list. To reorder or add/drop axes, insert `remap_axes` transforms in the subtransform list.

Invertibility: All ASDF tools are required to be able to compute the analytic inverse of this transform, by reversing the list of transforms and applying the inverse of each.

All of:

0

Type: transform-1.0.0 (page 58).

1

Type: any.

Examples:

A series of transforms:

```
!transform/compose-1.0.0
  forward:
    - !transform/generic-1.0.0
      n_inputs: 1
      n_outputs: 2
    - !transform/generic-1.0.0
      n_inputs: 2
      n_outputs: 1
```

Original schema in YAML

```
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/transform/compose-1.0.0"
tag: "tag:stsci.edu:asdf/transform/compose-1.0.0"
title: >
  Perform a list of subtransforms in series.

description: |
  The output of each subtransform is fed into the input of the next
  subtransform.

  The number of output dimensions of each subtransform must be equal
  to the number of input dimensions of the next subtransform in list.
  To reorder or add/drop axes, insert `remap_axes` transforms in the
  subtransform list.

  Invertibility: All ASDF tools are required to be able to compute the
  analytic inverse of this transform, by reversing the list of
  transforms and applying the inverse of each.

examples:
  -
    - A series of transforms
    - |
      !transform/compose-1.0.0
        forward:
          - !transform/generic-1.0.0
            n_inputs: 1
            n_outputs: 2
          - !transform/generic-1.0.0
            n_inputs: 2
            n_outputs: 1

allOf:
  - $ref: "transform-1.0.0"
  - properties:
      forward:
        type: array
        items:
          $ref: "transform-1.0.0"
    required: [forward]
```

concatenate: Send axes to different subtransforms.

Type: transform-1.0.0 (page 58) and any.

Send axes to different subtransforms.

Transforms a set of separable inputs by splitting the axes apart, sending them through the given subtransforms in parallel, and finally concatenating the subtransform output axes back together.

The input axes are assigned to each subtransform in order. If the number of input axes is unequal to the sum of the number of input axes of all of the subtransforms, that is considered an error case.

The output axes from each subtransform are appended together to make up the resulting output axes.

For example, given 5 input axes, and 3 subtransforms with the following orders:

1. transform A: 2 in -> 2 out
2. transform B: 1 in -> 2 out
3. transform C: 2 in -> 1 out

The transform is performed as follows:

```
:    i0    i1       i2       i3    i4
:    |     |        |        |     |
:  +---------+ +---------+ +----------+
:  |    A    | |    B    | |    C     |
:  +---------+ +---------+ +----------+
:    |     |     |     |         |
:    o0    o1    o2    o3        o4
```

If reordering of the input or output axes is required, use in series with the `remap_axes` transform.

Invertibility: All ASDF tools are required to be able to compute the analytic inverse of this transform.

All of:

0

Type: transform-1.0.0 (page 58).

1

Type: any.

Examples:

The example in the description:

```
!transform/concatenate-1.0.0
  forward:
    - !transform/generic-1.0.0
      n_inputs: 2
      n_outputs: 2
    - !transform/generic-1.0.0
      n_inputs: 1
      n_outputs: 2
    - !transform/generic-1.0.0
      n_inputs: 2
      n_outputs: 1
```

Original schema in YAML

```
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/transform/concatenate-1.0.0"
tag: "tag:stsci.edu:asdf/transform/concatenate-1.0.0"
title: >
  Send axes to different subtransforms.

description: |
  Transforms a set of separable inputs by splitting the axes apart,
  sending them through the given subtransforms in parallel, and
  finally concatenating the subtransform output axes back together.

  The input axes are assigned to each subtransform in order.  If the
  number of input axes is unequal to the sum of the number of input
  axes of all of the subtransforms, that is considered an error case.

  The output axes from each subtransform are appended together to make
  up the resulting output axes.

  For example, given 5 input axes, and 3 subtransforms with the
  following orders:

  1. transform A: 2 in -> 2 out
  1. transform B: 1 in -> 2 out
  1. transform C: 2 in -> 1 out

  The transform is performed as follows:

  ```
    :    i0    i1       i2       i3    i4
    :    |     |        |        |     |
    :  +---------+ +---------+ +----------+
    :  |    A    | |    B    | |    C     |
    :  +---------+ +---------+ +----------+
    :    |     |     |     |        |
    :    o0    o1    o2    o3       o4
  ```

  If reordering of the input or output axes is required, use in series
  with the `remap_axes` transform.

  Invertibility: All ASDF tools are required to be able to compute the
  analytic inverse of this transform.
examples:
  -
    - The example in the description
    - |
      !transform/concatenate-1.0.0
        forward:
          - !transform/generic-1.0.0
            n_inputs: 2
            n_outputs: 2
          - !transform/generic-1.0.0
            n_inputs: 1
            n_outputs: 2
```

```
        - !transform/generic-1.0.0
          n_inputs: 2
          n_outputs: 1

allOf:
  - $ref: "transform-1.0.0"
  - properties:
      forward:
        type: array
        items:
          $ref: "transform-1.0.0"
    required: [forward]
```

remap_axes: Reorder, add and drop axes.

Type: transform-1.0.0 (page 58) and any.

Reorder, add and drop axes.

This transform allows the order of the input axes to be shuffled and returned as the output axes.

It is a list made up of integers or "constant markers". Each item in the list corresponds to an output axis. For each item:

- If an integer, it is the index of the input axis to send to the output axis.
- If a constant, it must be a single item which is a constant value to send to the output axis.

If only a list is provided, the number of input axes is automatically determined from the maximum index in the list. If an object with `mapping` and `n_inputs` properties is provided, the number of input axes is explicitly set by the `n_inputs` value.

Invertibility: TBD

Definitions:

`mapping`

Type: array of (integer or constant-1.0.0 (page 38)).

Items:

Type: integer or constant-1.0.0 (page 38).

Any of:

Type: integer.

Type: constant-1.0.0 (page 38).

All of:

0

Type: transform-1.0.0 (page 58).

1

Type: any.

Examples:

For 2 input axes, swap the axes:

```
!transform/remap_axes-1.0.0
  mapping: [1, 0]
```

For 2 input axes, return the second axis and drop the first:

```
!transform/remap_axes-1.0.0
  mapping: [1]
```

For 2 input axes, return the first axis twice, followed by the second:

```
!transform/remap_axes-1.0.0
  mapping: [0, 0, 1]
```

For 2 input axes, add a third axis which is a constant:

```
!transform/remap_axes-1.0.0
  mapping: [0, 1, !core/constant-1.0.0 42]
```

The above example is equivalent to the following, and ASDF implementations are free to normalize it thusly:

```
!transform/concatenate-1.0.0
  forward:
    - !transform/remap_axes-1.0.0
      mapping: [0]
    - !transform/remap_axes-1.0.0
      mapping: [1]
    - !transform/constant-1.0.0
      value: 42
```

Here we have 3 input axes, but we are explicitly dropping the last one:

```
!transform/remap_axes-1.0.0
  mapping: [0, 1]
  n_inputs: 3
```

Original schema in YAML

```
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/transform/remap_axes-1.0.0"
tag: "tag:stsci.edu:asdf/transform/remap_axes-1.0.0"
title: >
  Reorder, add and drop axes.

description: |
  This transform allows the order of the input axes to be shuffled and
  returned as the output axes.

  It is a list made up of integers or "constant markers".  Each item
  in the list corresponds to an output axis.  For each item:

  - If an integer, it is the index of the input axis to send to the
```

```
    output axis.

  - If a constant, it must be a single item which is a constant value
    to send to the output axis.

  If only a list is provided, the number of input axes is
  automatically determined from the maximum index in the list.  If an
  object with `mapping` and `n_inputs` properties is provided, the
  number of input axes is explicitly set by the `n_inputs` value.

  Invertibility: TBD
examples:
  -
    - For 2 input axes, swap the axes
    - |
        !transform/remap_axes-1.0.0
          mapping: [1, 0]
  -
    - For 2 input axes, return the second axis and drop the first
    - |
        !transform/remap_axes-1.0.0
          mapping: [1]

  -
    - For 2 input axes, return the first axis twice, followed by the second
    - |
        !transform/remap_axes-1.0.0
          mapping: [0, 0, 1]

  -
    - For 2 input axes, add a third axis which is a constant
    - |
        !transform/remap_axes-1.0.0
          mapping: [0, 1, !core/constant-1.0.0 42]

  -
    - |
        The above example is equivalent to the following, and ASDF
        implementations are free to normalize it thusly:
    - |
        !transform/concatenate-1.0.0
          forward:
            - !transform/remap_axes-1.0.0
              mapping: [0]
            - !transform/remap_axes-1.0.0
              mapping: [1]
            - !transform/constant-1.0.0
              value: 42

  -
    - Here we have 3 input axes, but we are explicitly dropping the last one
    - |
        !transform/remap_axes-1.0.0
          mapping: [0, 1]
          n_inputs: 3

definitions:
  mapping:
```

```
    type: array
    items:
      anyOf:
        - type: integer
        - $ref: "../core/constant-1.0.0"

allOf:
  - $ref: "transform-1.0.0"
  - properties:
      n_inputs:
        description: |
          Explicitly set the number of input axes.  If not provided,
          it is determined from the maximum index value in the
          mapping list.
        type: integer
      mapping:
        $ref: "#/definitions/mapping"
    required: [mapping]
```

5.5.3 Arithmetic operations

add: Perform a list of subtransforms in parallel and then add their results together.

Type: transform-1.0.0 (page 58) and any.

Perform a list of subtransforms in parallel and then add their results together.

Each of the subtransforms must have the same number of inputs and outputs.

All of:

0

Type: transform-1.0.0 (page 58).

1

Type: any.

Examples:

A list of transforms, performed in parallel and added together:

```
!transform/add-1.0.0
  forward:
    - !transform/generic-1.0.0
      n_inputs: 1
      n_outputs: 2
    - !transform/generic-1.0.0
      n_inputs: 1
      n_outputs: 2
```

Original schema in YAML

```
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
```

```
id: "http://stsci.edu/schemas/asdf/transform/add-1.0.0"
tag: "tag:stsci.edu:asdf/transform/add-1.0.0"
title: >
  Perform a list of subtransforms in parallel and then
  add their results together.

description: |
  Each of the subtransforms must have the same number of inputs and
  outputs.

examples:
  -
    - A list of transforms, performed in parallel and added together
    - |
      !transform/add-1.0.0
        forward:
          - !transform/generic-1.0.0
            n_inputs: 1
            n_outputs: 2
          - !transform/generic-1.0.0
            n_inputs: 1
            n_outputs: 2

allOf:
  - $ref: "transform-1.0.0"
  - properties:
      forward:
        type: array
        items:
          $ref: "transform-1.0.0"
    required: [forward]
```

subtract: Perform a list of subtransforms in parallel and then subtract their results.

Type: transform-1.0.0 (page 58) and any.

Perform a list of subtransforms in parallel and then subtract their results.

Each of the subtransforms must have the same number of inputs and outputs.

Invertibility: This transform is not automatically invertible.

All of:

0

Type: transform-1.0.0 (page 58).

1

Type: any.

Examples:

A list of transforms, performed in parallel, and then combined through subtraction.:

```
!transform/subtract-1.0.0
  forward:
    - !transform/generic-1.0.0
      n_inputs: 1
```

```
    n_outputs: 2
  - !transform/generic-1.0.0
    n_inputs: 1
    n_outputs: 2
```

Original schema in YAML

```
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/transform/subtract-1.0.0"
tag: "tag:stsci.edu:asdf/transform/subtract-1.0.0"
title: >
  Perform a list of subtransforms in parallel and then
  subtract their results.

description: |
  Each of the subtransforms must have the same number of inputs and
  outputs.

  Invertibility: This transform is not automatically invertible.
examples:
  -
    - A list of transforms, performed in parallel, and then combined
      through subtraction.
    - |
      !transform/subtract-1.0.0
        forward:
          - !transform/generic-1.0.0
            n_inputs: 1
            n_outputs: 2
          - !transform/generic-1.0.0
            n_inputs: 1
            n_outputs: 2

allOf:
  - $ref: "transform-1.0.0"
  - properties:
      forward:
        type: array
        items:
          $ref: "transform-1.0.0"
    required: [forward]
```

multiply: Perform a list of subtransforms in parallel and then multiply their results.

Type: transform-1.0.0 (page 58) and any.

Perform a list of subtransforms in parallel and then multiply their results.

Each of the subtransforms must have the same number of inputs and outputs.

Invertibility: This transform is not automatically invertible.

All of:

`0`

Type: transform-1.0.0 (page 58).

`1`

Type: any.

Examples:

A list of transforms, performed in parallel, and then combined through multiplication.:

```
!transform/multiply-1.0.0
  forward:
    - !transform/generic-1.0.0
      n_inputs: 1
      n_outputs: 2
    - !transform/generic-1.0.0
      n_inputs: 1
      n_outputs: 2
```

Original schema in YAML

```
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/transform/multiply-1.0.0"
tag: "tag:stsci.edu:asdf/transform/multiply-1.0.0"
title: >
  Perform a list of subtransforms in parallel and then
  multiply their results.

description: |
  Each of the subtransforms must have the same number of inputs and
  outputs.

  Invertibility: This transform is not automatically invertible.
examples:
  -
    - A list of transforms, performed in parallel, and then combined
      through multiplication.
    - |
      !transform/multiply-1.0.0
        forward:
          - !transform/generic-1.0.0
            n_inputs: 1
            n_outputs: 2
          - !transform/generic-1.0.0
            n_inputs: 1
            n_outputs: 2

allOf:
  - $ref: "transform-1.0.0"
  - properties:
      forward:
        type: array
        items:
```

```
        $ref: "transform-1.0.0"
  required: [forward]
```

divide: Perform a list of subtransforms in parallel and then divide their results.

Type: transform-1.0.0 (page 58) and any.

Perform a list of subtransforms in parallel and then divide their results.

Each of the subtransforms must have the same number of inputs and outputs.

Invertibility: This transform is not automatically invertible.

All of:

0

Type: transform-1.0.0 (page 58).

1

Type: any.

Examples:

A list of transforms, performed in parallel, and then combined through division.:

```
!transform/divide-1.0.0
  forward:
    - !transform/generic-1.0.0
      n_inputs: 1
      n_outputs: 2
    - !transform/generic-1.0.0
      n_inputs: 1
      n_outputs: 2
```

Original schema in YAML

```
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/transform/divide-1.0.0"
tag: "tag:stsci.edu:asdf/transform/divide-1.0.0"
title: >
  Perform a list of subtransforms in parallel and then
  divide their results.

description: |
  Each of the subtransforms must have the same number of inputs and
  outputs.

  Invertibility: This transform is not automatically invertible.
examples:
  -
    - A list of transforms, performed in parallel, and then combined
      through division.
    - |
      !transform/divide-1.0.0
```

```
      forward:
        - !transform/generic-1.0.0
          n_inputs: 1
          n_outputs: 2
        - !transform/generic-1.0.0
          n_inputs: 1
          n_outputs: 2

allOf:
  - $ref: "transform-1.0.0"
  - properties:
      forward:
        type: array
        items:
          $ref: "transform-1.0.0"
    required: [forward]
```

power: Perform a list of subtransforms in parallel and then raise each result to the power of the next.

Type: transform-1.0.0 (page 58) and any.

Perform a list of subtransforms in parallel and then raise each result to the power of the next.

Each of the subtransforms must have the same number of inputs and outputs.

Invertibility: This transform is not automatically invertible.

All of:

0

Type: transform-1.0.0 (page 58).

1

Type: any.

Original schema in YAML

```
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/transform/power-1.0.0"
tag: "tag:stsci.edu:asdf/transform/power-1.0.0"
title: >
  Perform a list of subtransforms in parallel and then raise each
  result to the power of the next.

description: |
  Each of the subtransforms must have the same number of inputs and
  outputs.

  Invertibility: This transform is not automatically invertible.

allOf:
  - $ref: "transform-1.0.0"
  - properties:
      forward:
```

```
      type: array
      items:
        $ref: "transform-1.0.0"
  required: [forward]
```

5.5.4 Simple Transforms

shift: A Shift opeartion.

Type: object.

A Shift opeartion.

Apply an offset in one direction.

Properties:

`offset`

Type: number. Required.

Offset in one direction.

Original schema in YAML

```
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/transform/shift-1.0.0"
tag: "tag:stsci.edu:asdf/transform/shift-1.0.0"
title: >
  A Shift opeartion.
description: >
  Apply an offset in one direction.

type: object
properties:
  offset:
    type: number
    description: Offset in one direction.
required: [offset]
```

scale: A Scale model.

Type: object.

A Scale model.

Multiply the input by a factor.

Properties:

`factor`

Type: number. Required.

Multiplication factor.

Original schema in YAML

```yaml
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/transform/scale-1.0.0"
tag: "tag:stsci.edu:asdf/transform/scale-1.0.0"
title: >
  A Scale model.
description: >
  Multiply the input by a factor.

type: object
properties:
  factor:
    type: number
    description: Multiplication factor.
required: [factor]
```

5.5.5 Projections

Affine

affine: An affine transform.

Type: transform-1.0.0 (page 58) and object.

An affine transform.

Invertibility: All ASDF tools are required to be able to compute the analytic inverse of this transform.

All of:

`0`

Type: transform-1.0.0 (page 58).

`1`

Type: object.

Properties:

`matrix`

Type: ndarray-1.0.0 (page 20) or array of (array of (number) *len* = 2) *len* = 2. Required.

An array of size (n x n), where n is the number of axes, representing the linear transformation in an affine transform.

Any of:

`---`

Type: ndarray-1.0.0 (page 20).

`---`

Type: array of (array of (number) *len* = 2) *len* = 2.

Items:

Type: array of (number) *len* = 2.

Items:

Type: number.

`translation`

Type: ndarray-1.0.0 (page 20) or array of (number) *len* = 2.

An array of size (n,), where n is the number of axes, representing the translation in an affine transform.

Any of:

Type: ndarray-1.0.0 (page 20).

Type: array of (number) *len* = 2.

Items:

Type: number.

Original schema in YAML


```yaml
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/transform/affine-1.0.0"
tag: "tag:stsci.edu:asdf/transform/affine-1.0.0"
title: >
  An affine transform.
description: |
  Invertibility: All ASDF tools are required to be able to compute the
  analytic inverse of this transform.


allOf:
  - $ref: "transform-1.0.0"
  - type: object
    properties:
      matrix:
        description: |
          An array of size (*n* x *n*), where *n* is the number of axes,
          representing the linear transformation in an affine transform.
        anyOf:
          - $ref: "../core/ndarray-1.0.0"
          - type: array
            items:
              type: array
              items:
                type: number
              minItems: 2
              maxItems: 2
            minItems: 2
            maxItems: 2
      translation:
        description: |
          An array of size (*n*,), where  *n* is the number of axes,
          representing the translation in an affine transform.
```

```
      anyOf:
        - $ref: "../core/ndarray-1.0.0"
        - type: array
          items:
            type: number
          minItems: 2
          maxItems: 2
  required: [matrix]
```

rotate2d: A 2D rotation.

Type: transform-1.0.0 (page 58) and object.

A 2D rotation.

A 2D rotation around the origin, in degrees. Invertibility: All ASDF tools are required to be able to compute the analytic inverse of this transform.

All of:

0

Type: transform-1.0.0 (page 58).

1

Type: object.

Properties:

angle

Type: number. Required.

Angle, in degrees.

Original schema in YAML

```
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/transform/rotate2d-1.0.0"
tag: "tag:stsci.edu:asdf/transform/rotate2d-1.0.0"
title: >
  A 2D rotation.
description: >
  A 2D rotation around the origin, in degrees.

  Invertibility: All ASDF tools are required to be able to compute the
  analytic inverse of this transform.
allOf:
  - $ref: "transform-1.0.0"
  - type: object
    properties:
      angle:
        type: number
        description: Angle, in degrees.
    required: [angle]
```

rotate3d: Rotation in 3D space.

Type: transform-1.0.0 (page 58) and object.

Rotation in 3D space.

Euler angle rotation around 3 axes.

Invertibility: All ASDF tools are required to be able to compute the analytic inverse of this transform.

All of:

`0`

Type: transform-1.0.0 (page 58).

`1`

Type: object.

Properties:

`phi`

Type: number. Required.

Angle, in degrees.

`theta`

Type: number. Required.

Angle, in degrees.

`psi`

Type: number. Required.

Angle, in degrees.

`direction`

Type: any from [”zxz”, “zyz”, “yzy”, “yxy”, “xyx”, “xzx”, “native2celestial”, “celestial2native”]. Required.

Sequence of rotation axes: one of `zxz`, `zyz`, `yzy`, `yxy`, `xyx`, `xzx` or `native2celestial`, `celestial2native`.

If `direction` is `native2celestial` or `celestial2native`, `phi`, `theta` are the longitude and latitude of the native pole in the celestial system and `psi` is the longitude of the celestial pole in the native system.

Default: “native2celestial”

Examples:

The three Euler angles are 12.3, 34 and -1.2 in degrees.:

```
!transform/rotate3d-1.0.0
  phi: 12.3
  theta: 34
  psi: -1.2
  direction: zxz
```

Original schema in YAML

```yaml
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/transform/rotate3d-1.0.0"
tag: "tag:stsci.edu:asdf/transform/rotate3d-1.0.0"
title: >
  Rotation in 3D space.
description: |
  Euler angle rotation around 3 axes.

  Invertibility: All ASDF tools are required to be able to compute the
  analytic inverse of this transform.

examples:
  -
    - The three Euler angles are 12.3, 34 and -1.2 in degrees.
    - |
      !transform/rotate3d-1.0.0
        phi: 12.3
        theta: 34
        psi: -1.2
        direction: zxz

allOf:
  - $ref: "transform-1.0.0"
  - type: object
    properties:
      phi:
        type: number
        description: Angle, in degrees.
      theta:
        type: number
        description: Angle, in degrees.
      psi:
        type: number
        description: Angle, in degrees.
      direction:
        description: |
          Sequence of rotation axes: one of `zxz`, `zyz`, `yzy`, `yxy`, `xyx`, `xzx`
          or `native2celestial`, `celestial2native`.

          If `direction` is `native2celestial` or `celestial2native`,
          `phi`, `theta` are the longitude and latitude of the native pole in
          the celestial system and `psi` is the longitude of the celestial pole in
          the native system.

        enum: [zxz, zyz, yzy, yxy, xyx, xzx, native2celestial, celestial2native]
        default: native2celestial

    required: [phi, theta, psi, direction]
```

Zenithal (azimuthal)

zenithal: Base class of all zenithal (or azimuthal) projections.

Type: transform-1.0.0 (page 58) and object.

Base class of all zenithal (or azimuthal) projections.

Zenithal projections are completely specified by defining the radius as a function of native latitude, R_θ.

The pixel-to-sky transformation is defined as:

$$\begin{aligned}\phi &= \arg(-y, x) \\ R_\theta &= \sqrt{x^2 + y^2}\end{aligned}$$

and the inverse (sky-to-pixel) is defined as:

$$\begin{aligned}x &= R_\theta \sin \phi \\ y &= R_\theta \cos \phi\end{aligned}$$

All of:

`0`

Type: transform-1.0.0 (page 58).

`1`

Type: object.

Properties:

`direction`

Type: any from ["pix2sky", "sky2pix"].

Default: "pix2sky"

Original schema in YAML

```yaml
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/transform/zenithal-1.0.0"
title: |
  Base class of all zenithal (or azimuthal) projections.

description: |
  Zenithal projections are completely specified by defining the radius
  as a function of native latitude, $R_\theta$.

  The pixel-to-sky transformation is defined as:

  $$\phi &= \arg(-y, x) \\
  R_\theta &= \sqrt{x^2 + y^2}$$

  and the inverse (sky-to-pixel) is defined as:

  $$x &= R_\theta \sin \phi \\
  y &= R_\theta \cos \phi$$

allOf:
  - $ref: "transform-1.0.0"
  - type: object
    properties:
      direction:
        enum: [pix2sky, sky2pix]
        default: pix2sky
```

gnomonic: The gnomonic projection.

Type: zenithal-1.0.0 (page 81).

The gnomonic projection.

Corresponds to the TAN projection in the FITS WCS standard.

See *zenithal* (page 81) for the definition of the full transformation.

The pixel-to-sky transformation is defined as:

$$\theta = \tan^{-1}\left(\frac{180^{\circ}}{\pi R_\theta}\right)$$

And the sky-to-pixel transformation is defined as:

$$R_\theta = \frac{180^{\circ}}{\pi}\cot \theta$$

Invertibility: All ASDF tools are required to provide the inverse of this transform.

Original schema in YAML

```yaml
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/transform/gnomonic-1.0.0"
tag: "tag:stsci.edu:asdf/transform/gnomonic-1.0.0"
title: |
  The gnomonic projection.

description: |
  Corresponds to the `TAN` projection in the FITS WCS standard.

  See
  [zenithal](ref:http://stsci.edu/schemas/asdf/transform/zenithal-1.0.0)
  for the definition of the full transformation.

  The pixel-to-sky transformation is defined as:

  $$\theta = \tan^{-1}\left(\frac{180^{\circ}}{\pi R_\theta}\right)$$

  And the sky-to-pixel transformation is defined as:

  $$R_\theta = \frac{180^{\circ}}{\pi}\cot \theta$$

  Invertibility: All ASDF tools are required to provide the inverse of
  this transform.

$ref: "zenithal-1.0.0"
```

zenithal_perspective: The zenithal perspective projection.

Type: zenithal-1.0.0 (page 81) and object.

The zenithal perspective projection.

Corresponds to the AZP projection in the FITS WCS standard.

The pixel-to-sky transformation is defined as:

$$\phi = \arg(-y \cos\gamma, x)$$
$$\theta = \begin{cases} \psi - \omega \\ \psi + \omega + 180^\circ \end{cases}$$

where:

$$\psi = \arg(\rho, 1)$$
$$\omega = \sin^{-1}\left(\frac{\rho\mu}{\sqrt{\rho^2+1}}\right)$$
$$\rho = \frac{R}{\frac{180^\circ}{\pi}(\mu+1) + y\sin\gamma}$$
$$R = \sqrt{x^2 + y^2\cos^2\gamma}$$

And the sky-to-pixel transformation is defined as:

$$x = R\sin\phi$$
$$y = -R\sec\gamma\cos\theta$$

where:

$$R = \frac{180^\circ}{\pi}\frac{(\mu+1)\cos\theta}{(\mu+\sin\theta)+\cos\theta\cos\phi\tan\gamma}$$

Invertibility: All ASDF tools are required to provide the inverse of this transform.

All of:

`0`

Type: zenithal-1.0.0 (page 81).

`1`

Type: object.

Properties:

`mu`

Type: number.

Distance from point of projection to center of sphere in spherical radii.

Default: 0

`gamma`

Type: number.

Look angle, in degrees.

Default: 0

Original schema in YAML

```
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/transform/zenithal_perspective-1.0.0"
```

```
tag: "tag:stsci.edu:asdf/transform/zenithal_perspective-1.0.0"
title: |
  The zenithal perspective projection.

description: |
  Corresponds to the `AZP` projection in the FITS WCS standard.

  The pixel-to-sky transformation is defined as:

  $$\phi &= \arg(-y \cos \gamma, x) \\
  \theta &= \left\{\genfrac{}{}{0pt}{}{\psi - \omega}{\psi + \omega + 180^{\circ}}\right.$$

  where:

  $$\psi &= \arg(\rho, 1) \\
  \omega &= \sin^{-1}\left(\frac{\rho \mu}{\sqrt{\rho^2 + 1}}\right) \\
  \rho &= \frac{R}{\frac{180^{\circ}}{\pi}(\mu + 1) + y \sin \gamma} \\
  R &= \sqrt{x^2 + y^2 \cos^2 \gamma}$$

  And the sky-to-pixel transformation is defined as:

  $$x &= R \sin \phi \\
  y &= -R \sec \gamma \cos \theta$$

  where:

  $$R = \frac{180^{\circ}}{\pi} \frac{(\mu + 1) \cos \theta}{(\mu + \sin \theta) + \cos \theta \cos \phi \tan \gamma}$$

  Invertibility: All ASDF tools are required to provide the inverse of
  this transform.

allOf:
  - $ref: "zenithal-1.0.0"
  - type: object
    properties:
      mu:
        type: number
        description: |
          Distance from point of projection to center of sphere in
          spherical radii.
        default: 0

      gamma:
        type: number
        description: |
          Look angle, in degrees.
        default: 0
```

slant_zenithal_perspective: The slant zenithal perspective projection.

Type: zenithal-1.0.0 (page 81) and object.

The slant zenithal perspective projection.

Corresponds to the SZP projection in the FITS WCS standard.

See *zenithal* (page 81) for the definition of the full transformation.

The pixel-to-sky transformation is defined as:

$$\theta = \tan^{-1}\left(\frac{180^\circ}{\pi R_\theta}\right)$$

And the sky-to-pixel transformation is defined as:

$$R_\theta = \frac{180^\circ}{\pi} \cot \theta$$

Invertibility: All ASDF tools are required to provide the inverse of this transform.

All of:

`0`

Type: zenithal-1.0.0 (page 81).

`1`

Type: object.

Properties:

`mu`

Type: number.

Distance from point of projection to center of sphere in spherical radii.

Default: 0

`phi0`

Type: number.

The longitude ϕ_0 of the reference point, in degrees.

Default: 0

`theta0`

Type: number.

The latitude θ_0 of the reference point, in degrees.

Default: 90

Original schema in YAML

```
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/transform/slant_zenithal_perspective-1.0.0"
tag: "tag:stsci.edu:asdf/transform/slant_zenithal_perspective-1.0.0"
title: |
  The slant zenithal perspective projection.

description: |
  Corresponds to the `SZP` projection in the FITS WCS standard.

  See
  [zenithal](ref:http://stsci.edu/schemas/asdf/transform/zenithal-1.0.0)
  for the definition of the full transformation.

```

```
  The pixel-to-sky transformation is defined as:

  $$\theta = \tan^{-1}\left(\frac{180^{\circ}}{\pi R_\theta}\right)$$

  And the sky-to-pixel transformation is defined as:

  $$R_\theta = \frac{180^{\circ}}{\pi}\cot \theta$$

  Invertibility: All ASDF tools are required to provide the inverse of
  this transform.

allOf:
  - $ref: "zenithal-1.0.0"
  - type: object
    properties:
      mu:
        type: number
        description: |
          Distance from point of projection to center of sphere in
          spherical radii.
        default: 0

      phi0:
        type: number
        description: |
          The longitude $\phi_0$ of the reference point, in degrees.
        default: 0

      theta0:
        type: number
        description: |
          The latitude $\theta_0$ of the reference point, in degrees.
        default: 90
```

stereographic: The stereographic projection.

Type: zenithal-1.0.0 (page 81).

The stereographic projection.

Corresponds to the STG projection in the FITS WCS standard.

See *zenithal* (page 81) for the definition of the full transformation.

The pixel-to-sky transformation is defined as:

$$\theta = 90^\circ - 2\tan^{-1}\left(\frac{\pi R_\theta}{360^\circ}\right)$$

And the sky-to-pixel transformation is defined as:

$$R_\theta = \frac{180^\circ}{\pi}\frac{2\cos\theta}{1+\sin\theta}$$

Invertibility: All ASDF tools are required to provide the inverse of this transform.

Original schema in YAML

```yaml
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/transform/stereographic-1.0.0"
tag: "tag:stsci.edu:asdf/transform/stereographic-1.0.0"
title: |
  The stereographic projection.

description: |
  Corresponds to the `STG` projection in the FITS WCS standard.

  See
  [zenithal](ref:http://stsci.edu/schemas/asdf/transform/zenithal-1.0.0)
  for the definition of the full transformation.

  The pixel-to-sky transformation is defined as:

  $$\theta = 90^{\circ} - 2 \tan^{-1}\left(\frac{\pi R_\theta}{360^{\circ}}\right)$$

  And the sky-to-pixel transformation is defined as:

  $$R_\theta = \frac{180^{\circ}}{\pi}\frac{2 \cos \theta}{1 + \sin \theta}$$

  Invertibility: All ASDF tools are required to provide the inverse of
  this transform.

$ref: "zenithal-1.0.0"
```

slant_orthographic: The slant orthographic projection.

Type: zenithal-1.0.0 (page 81).

The slant orthographic projection.

Corresponds to the `SIN` projection in the FITS WCS standard.

See *zenithal* (page 81) for the definition of the full transformation.

The pixel-to-sky transformation is defined as:

$$\theta = \cos^{-1}\left(\frac{\pi}{180^{\circ}}R_\theta\right)$$

And the sky-to-pixel transformation is defined as:

$$R_\theta = \frac{180^{\circ}}{\pi}\cos\theta$$

Invertibility: All ASDF tools are required to provide the inverse of this transform.

Original schema in YAML

```yaml
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/transform/slant_orthographic-1.0.0"
tag: "tag:stsci.edu:asdf/transform/slant_orthographic-1.0.0"
title: |
```

```yaml
  The slant orthographic projection.

description: |
  Corresponds to the `SIN` projection in the FITS WCS standard.

  See
  [zenithal](ref:http://stsci.edu/schemas/asdf/transform/zenithal-1.0.0)
  for the definition of the full transformation.

  The pixel-to-sky transformation is defined as:

  $$\theta = \cos^{-1}\left(\frac{\pi}{180^{\circ}}R_\theta\right)$$

  And the sky-to-pixel transformation is defined as:

  $$R_\theta = \frac{180^{\circ}}{\pi}\cos \theta$$

  Invertibility: All ASDF tools are required to provide the inverse of
  this transform.

$ref: "zenithal-1.0.0"
```

zenithal_equidistant: The zenithal equidistant projection.

Type: zenithal-1.0.0 (page 81).

The zenithal equidistant projection.

Corresponds to the ARC projection in the FITS WCS standard.

See *zenithal* (page 81) for the definition of the full transformation.

The pixel-to-sky transformation is defined as:

$$\theta = 90^{\circ} - R_\theta$$

And the sky-to-pixel transformation is defined as:

$$R_\theta = 90^{\circ} - \theta$$

Invertibility: All ASDF tools are required to provide the inverse of this transform.

Original schema in YAML

```yaml
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/transform/zenithal_equidistant-1.0.0"
tag: "tag:stsci.edu:asdf/transform/zenithal_equidistant-1.0.0"
title: |
  The zenithal equidistant projection.

description: |
  Corresponds to the `ARC` projection in the FITS WCS standard.

  See
  [zenithal](ref:http://stsci.edu/schemas/asdf/transform/zenithal-1.0.0)
  for the definition of the full transformation.
```

```
  The pixel-to-sky transformation is defined as:

  $$\theta = 90^\circ - R_\theta$$

  And the sky-to-pixel transformation is defined as:

  $$R_\theta = 90^\circ - \theta$$

  Invertibility: All ASDF tools are required to provide the inverse of
  this transform.

$ref: "zenithal-1.0.0"
```

zenithal_equal_area: The zenithal equal area projection.

Type: zenithal-1.0.0 (page 81).

The zenithal equal area projection.

Corresponds to the ZEA projection in the FITS WCS standard.

See *zenithal* (page 81) for the definition of the full transformation.

The pixel-to-sky transformation is defined as:

$$\theta = 90^\circ - 2\sin^{-1}\left(\frac{\pi R_\theta}{360^\circ}\right)$$

And the sky-to-pixel transformation is defined as:

$$\begin{aligned} R_\theta &= \frac{180^\circ}{\pi}\sqrt{2(1-\sin\theta)} \\ &= \frac{360^\circ}{\pi}\sin\left(\frac{90^\circ-\theta}{2}\right) \end{aligned}$$

Invertibility: All ASDF tools are required to provide the inverse of this transform.

Original schema in YAML

```
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/transform/zenithal_equal_area-1.0.0"
tag: "tag:stsci.edu:asdf/transform/zenithal_equal_area-1.0.0"
title: |
  The zenithal equal area projection.

description: |
  Corresponds to the `ZEA` projection in the FITS WCS standard.

  See
  [zenithal](ref:http://stsci.edu/schemas/asdf/transform/zenithal-1.0.0)
  for the definition of the full transformation.

  The pixel-to-sky transformation is defined as:

  $$\theta = 90^\circ - 2 \sin^{-1} \left(\frac{\pi R_\theta}{360^\circ}\right)$$
```

```

  And the sky-to-pixel transformation is defined as:

  $$R_\theta &= \frac{180^\circ}{\pi} \sqrt{2(1 - \sin\theta)} \\
             &= \frac{360^\circ}{\pi} \sin\left(\frac{90^\circ - \theta}{2}\right)$$

  Invertibility: All ASDF tools are required to provide the inverse of
  this transform.

$ref: "zenithal-1.0.0"
```

airy: The Airy projection.

Type: zenithal-1.0.0 (page 81) and object.

The Airy projection.

Corresponds to the `AIR` projection in the FITS WCS standard.

See *zenithal* (page 81) for the definition of the full transformation.

All of:

`0`

Type: zenithal-1.0.0 (page 81).

`1`

Type: object.

Properties:

`theta_b`

Type: number.

The latitude θ_b at which to minimize the error, in degrees.

Default: 90

Original schema in YAML

```yaml
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/transform/airy-1.0.0"
tag: "tag:stsci.edu:asdf/transform/airy-1.0.0"
title: |
  The Airy projection.

description: |
  Corresponds to the `AIR` projection in the FITS WCS standard.

  See
  [zenithal](ref:http://stsci.edu/schemas/asdf/transform/zenithal-1.0.0)
  for the definition of the full transformation.

allOf:
  - $ref: "zenithal-1.0.0"
  - type: object
```

```
    properties:
      theta_b:
        type: number
        description: |
          The latitude $\theta_b$ at which to minimize the error, in
          degrees.
        default: 90
```

Cylindrical

cylindrical: Base class of all cylindrical projections.

Type: transform-1.0.0 (page 58) and object.

Base class of all cylindrical projections.

The surface of cylindrical projections is a cylinder.

All of:

`0`

Type: transform-1.0.0 (page 58).

`1`

Type: object.

Properties:

`direction`

Type: any from ["pix2sky", "sky2pix"].

Default: "pix2sky"

Original schema in YAML

```
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/transform/cylindrical-1.0.0"
title: |
  Base class of all cylindrical projections.

description: |
  The surface of cylindrical projections is a cylinder.

allOf:
  - $ref: "transform-1.0.0"
  - type: object
    properties:
      direction:
        enum: [pix2sky, sky2pix]
        default: pix2sky
```

cylindrical_perspective: The cylindrical perspective projection.

Type: cylindrical-1.0.0 (page 92) and object.

The cylindrical perspective projection.

Corresponds to the `CYP` projection in the FITS WCS standard.

The pixel-to-sky transformation is defined as:

$$\phi = \frac{x}{\lambda}$$
$$\theta = \arg(1, \eta) + \sin{-1}\left(\frac{\eta\mu}{\sqrt{\eta^2 + 1}}\right)$$

And the sky-to-pixel transformation is defined as:

$$x = \lambda\phi$$
$$y = \frac{180^\circ}{\pi}\left(\frac{\mu + \lambda}{\mu + \cos\theta}\right)\sin\theta$$

Invertibility: All ASDF tools are required to provide the inverse of this transform.

All of:

`0`

Type: cylindrical-1.0.0 (page 92).

`1`

Type: object.

Properties:

`mu`

Type: number.

Distance from center of sphere in the direction opposite the projected surface, in spherical radii.

Default: 0

`lambda`

Type: number.

Radius of the cylinder in spherical radii, default is 0.

Default: 0

Original schema in YAML

```yaml
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/transform/cylindrical_perspective-1.0.0"
tag: "tag:stsci.edu:asdf/transform/cylindrical_perspective-1.0.0"
title: |
  The cylindrical perspective projection.

description: |
  Corresponds to the `CYP` projection in the FITS WCS standard.
```

```

  The pixel-to-sky transformation is defined as:

  $$\phi &= \frac{x}{\lambda} \\
  \theta &= \arg(1, \eta) + \sin{-1}\left(\frac{\eta \mu}{\sqrt{\eta^2 + 1}}\right)$$

  And the sky-to-pixel transformation is defined as:

  $$x &= \lambda \phi \\
  y &= \frac{180^{\circ}}{\pi}\left(\frac{\mu + \lambda}{\mu + \cos \theta}\right)\sin \theta$$

  Invertibility: All ASDF tools are required to provide the inverse of
  this transform.

allOf:
  - $ref: "cylindrical-1.0.0"
  - type: object
    properties:
      mu:
        type: number
        description: |
          Distance from center of sphere in the direction opposite the
          projected surface, in spherical radii.
        default: 0

      lambda:
        type: number
        description: |
          Radius of the cylinder in spherical radii, default is 0.
        default: 0
```

cylindrical_equal_area: The cylindrical equal area projection.

Type: cylindrical-1.0.0 (page 92) and object.

The cylindrical equal area projection.

Corresponds to the CEA projection in the FITS WCS standard.

The pixel-to-sky transformation is defined as:

$$\phi = x$$
$$\theta = \sin^{-1}\left(\frac{\pi}{180^\circ}\lambda y\right)$$

And the sky-to-pixel transformation is defined as:

$$x = \phi$$
$$y = \frac{180^\circ}{\pi}\frac{\sin\theta}{\lambda}$$

Invertibility: All ASDF tools are required to provide the inverse of this transform.

All of:

0

Type: cylindrical-1.0.0 (page 92).

1

Type: object.

Properties:

`lambda`

Type: number.

Radius of the cylinder in spherical radii, default is 0.

Default: 0

Original schema in YAML

```yaml
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/transform/cylindrical_equal_area-1.0.0"
tag: "tag:stsci.edu:asdf/transform/cylindrical_equal_area-1.0.0"
title: |
  The cylindrical equal area projection.

description: |
  Corresponds to the `CEA` projection in the FITS WCS standard.

  The pixel-to-sky transformation is defined as:

  $$\phi &= x \\
  \theta &= \sin^{-1}\left(\frac{\pi}{180^{\circ}}\lambda y\right)$$

  And the sky-to-pixel transformation is defined as:

  $$x &= \phi \\
  y &= \frac{180^{\circ}}{\pi}\frac{\sin \theta}{\lambda}$$

  Invertibility: All ASDF tools are required to provide the inverse of
  this transform.

allOf:
  - $ref: "cylindrical-1.0.0"
  - type: object
    properties:
      lambda:
        type: number
        description: |
          Radius of the cylinder in spherical radii, default is 0.
        default: 0
```

plate_carree: The plate carrée projection.

Type: cylindrical-1.0.0 (page 92).

The plate carrée projection.

Corresponds to the `CAR` projection in the FITS WCS standard.

The main virtue of this transformation is its simplicity.

The pixel-to-sky transformation is defined as:

$$\phi = x$$
$$\theta = y$$

And the sky-to-pixel transformation is defined as:

$$x = \phi$$
$$y = \theta$$

Invertibility: All ASDF tools are required to provide the inverse of this transform.

Original schema in YAML

```yaml
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/transform/plate_carree-1.0.0"
tag: "tag:stsci.edu:asdf/transform/plate_carree-1.0.0"
title: |
  The plate carrée projection.

description: |
  Corresponds to the `CAR` projection in the FITS WCS standard.

  The main virtue of this transformation is its simplicity.

  The pixel-to-sky transformation is defined as:

  $$\phi &= x \\
  \theta &= y$$

  And the sky-to-pixel transformation is defined as:

  $$x &= \phi \\
  y &= \theta$$

  Invertibility: All ASDF tools are required to provide the inverse of
  this transform.

$ref: "cylindrical-1.0.0"
```

mercator: The Mercator projection.

Type: cylindrical-1.0.0 (page 92).

The Mercator projection.

Corresponds to the `MER` projection in the FITS WCS standard.

The pixel-to-sky transformation is defined as:

$$\phi = x$$
$$\theta = 2\tan^{-1}\left(e^{y\pi/180^\circ}\right) - 90^\circ$$

And the sky-to-pixel transformation is defined as:

$$x = \phi$$
$$y = \frac{180^{\circ}}{\pi}\ln\tan\left(\frac{90^{\circ} + \theta}{2}\right)$$

Invertibility: All ASDF tools are required to provide the inverse of this transform.

Original schema in YAML

```yaml
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/transform/mercator-1.0.0"
tag: "tag:stsci.edu:asdf/transform/mercator-1.0.0"
title: |
  The Mercator projection.

description: |
  Corresponds to the `MER` projection in the FITS WCS standard.

  The pixel-to-sky transformation is defined as:

  $$\phi &= x \\
  \theta &= 2 \tan^{-1}\left(e^{y \pi / 180^{\circ}}\right)-90^{\circ}$$

  And the sky-to-pixel transformation is defined as:

  $$x &= \phi \\
  y &= \frac{180^{\circ}}{\pi}\ln \tan \left(\frac{90^{\circ} + \theta}{2}\right)$$

  Invertibility: All ASDF tools are required to provide the inverse of
  this transform.

$ref: "cylindrical-1.0.0"
```

Pseudocylindrical

pseudocylindrical: Base class of all pseudocylindrical projections.

Type: transform-1.0.0 (page 58) and object.

Base class of all pseudocylindrical projections.

Pseudocylindrical projections are like cylindrical projections except the parallels of latitude are projected at diminishing lengths toward the polar regions in order to reduce lateral distortion there. Consequently, the meridians are curved.

All of:

0

Type: transform-1.0.0 (page 58).

1

Type: object.

Properties:

direction

Type: any from ["pix2sky", "sky2pix"].

Default: "pix2sky"

Original schema in YAML


```yaml
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/transform/pseudocylindrical-1.0.0"
title: |
  Base class of all pseudocylindrical projections.

description: |
  Pseudocylindrical projections are like cylindrical projections
  except the parallels of latitude are projected at diminishing
  lengths toward the polar regions in order to reduce lateral
  distortion there.  Consequently, the meridians are curved.

allOf:
  - $ref: "transform-1.0.0"
  - type: object
    properties:
      direction:
        enum: [pix2sky, sky2pix]
        default: pix2sky
```


sanson_flamsteed: The Sanson-Flamsteed projection.

Type: pseudocylindrical-1.0.0 (page 97).

The Sanson-Flamsteed projection.

Corresponds to the SFL projection in the FITS WCS standard.

The pixel-to-sky transformation is defined as:

$$\phi = \frac{x}{\cos y}$$
$$\theta = y$$

And the sky-to-pixel transformation is defined as:

$$x = \phi \cos \theta$$
$$y = \theta$$

Invertibility: All ASDF tools are required to provide the inverse of this transform.

Original schema in YAML


```yaml
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/transform/sanson_flamsteed-1.0.0"
tag: "tag:stsci.edu:asdf/transform/sanson_flamsteed-1.0.0"
title: |
```

```
  The Sanson-Flamsteed projection.

description: |
  Corresponds to the `SFL` projection in the FITS WCS standard.

  The pixel-to-sky transformation is defined as:

  $$\phi &= \frac{x}{\cos y} \\
    \theta &= y$$

  And the sky-to-pixel transformation is defined as:

  $$x &= \phi \cos \theta \\
    y &= \theta$$

  Invertibility: All ASDF tools are required to provide the inverse of
  this transform.

$ref: "pseudocylindrical-1.0.0"
```

parabolic: Parabolic projection.

Type: pseudocylindrical-1.0.0 (page 97).

Parabolic projection.

Corresponds to the PAR projection in the FITS WCS standard.

The pixel-to-sky transformation is defined as:

$$\phi = \frac{180^\circ}{\pi} \frac{x}{1 - 4(y/180^\circ)^2}$$
$$\theta = 3 \sin^{-1}\left(\frac{y}{180^\circ}\right)$$

And the sky-to-pixel transformation is defined as:

$$x = \phi\left(2\cos\frac{2\theta}{3} - 1\right)$$
$$y = 180^\circ \sin\frac{\theta}{3}$$

Invertibility: All ASDF tools are required to provide the inverse of this transform.

Original schema in YAML

```
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/transform/parabolic-1.0.0"
tag: "tag:stsci.edu:asdf/transform/parabolic-1.0.0"
title: |
  Parabolic projection.

description: |
  Corresponds to the `PAR` projection in the FITS WCS standard.

```

```
  The pixel-to-sky transformation is defined as:

  $$\phi &= \frac{180^\circ}{\pi} \frac{x}{1 - 4(y / 180^\circ)^2} \\
    \theta &= 3 \sin^{-1}\left(\frac{y}{180^\circ}\right)$$

  And the sky-to-pixel transformation is defined as:

  $$x &= \phi \left(2\cos\frac{2\theta}{3} - 1\right) \\
    y &= 180^\circ \sin \frac{\theta}{3}$$

  Invertibility: All ASDF tools are required to provide the inverse of
  this transform.

$ref: "pseudocylindrical-1.0.0"
```

molleweide: Molleweide's projection.

Type: pseudocylindrical-1.0.0 (page 97).

Molleweide's projection.

Corresponds to the `MOL` projection in the FITS WCS standard.

The pixel-to-sky transformation is defined as:

$$\phi = \frac{\pi x}{2\sqrt{2-\left(\frac{\pi}{180^\circ}y\right)^2}}$$

$$\theta = \sin^{-1}\left(\frac{1}{90^\circ}\sin^{-1}\left(\frac{\pi}{180^\circ}\frac{y}{\sqrt{2}}\right)+\frac{y}{180^\circ}\sqrt{2-\left(\frac{\pi}{180^\circ}y\right)^2}\right)$$

And the sky-to-pixel transformation is defined as:

$$x = \frac{2\sqrt{2}}{\pi}\phi\cos\gamma$$

$$y = \sqrt{2}\frac{180^\circ}{\pi}\sin\gamma$$

Invertibility: All ASDF tools are required to provide the inverse of this transform.

Original schema in YAML

```
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/transform/molleweide-1.0.0"
tag: "tag:stsci.edu:asdf/transform/molleweide-1.0.0"
title: |
  Molleweide's projection.

description: |
  Corresponds to the `MOL` projection in the FITS WCS standard.

  The pixel-to-sky transformation is defined as:

  $$\phi &= \frac{\pi x}{2 \sqrt{2 - \left(\frac{\pi}{180^\circ}y\right)^2}} \\
    \theta &= \sin^{-1}\left(\frac{1}{90^\circ}\sin^{-1}\left(\frac{\pi}{180^\circ}\frac{y}{\sqrt{2}}\right) + \frac{y}{
```

```

  And the sky-to-pixel transformation is defined as:

  $$x &= \frac{2 \sqrt{2}}{\pi} \phi \cos \gamma \\
    y &= \sqrt{2} \frac{180^\circ}{\pi} \sin \gamma$$

  Invertibility: All ASDF tools are required to provide the inverse of
  this transform.

$ref: "pseudocylindrical-1.0.0"
```

hammer_aitoff: Hammer-Aitoff projection.

Type: pseudocylindrical-1.0.0 (page 97).

Hammer-Aitoff projection.

Corresponds to the `AIT` projection in the FITS WCS standard.

The pixel-to-sky transformation is defined as:

$$\phi = 2 \arg \left(2Z^2 - 1, \frac{\pi}{180^\circ} \frac{Z}{2}x\right)$$
$$\theta = \sin^{-1}\left(\frac{\pi}{180^\circ}yZ\right)$$

And the sky-to-pixel transformation is defined as:

$$x = 2\gamma \cos \theta \sin \frac{\phi}{2}$$
$$y = \gamma \sin \theta$$

where:

$$\gamma = \frac{180^\circ}{\pi} \sqrt{\frac{2}{1 + \cos \theta \cos(\phi / 2)}}$$

Invertibility: All ASDF tools are required to provide the inverse of this transform.

Original schema in YAML

```
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/transform/hammer_aitoff-1.0.0"
tag: "tag:stsci.edu:asdf/transform/hammer_aitoff-1.0.0"
title: |
  Hammer-Aitoff projection.

description: |
  Corresponds to the `AIT` projection in the FITS WCS standard.

  The pixel-to-sky transformation is defined as:

  $$\phi &= 2 \arg \left(2Z^2 - 1, \frac{\pi}{180^\circ} \frac{Z}{2}x\right) \\
    \theta &= \sin^{-1}\left(\frac{\pi}{180^\circ}yZ\right)$$

```

```
  And the sky-to-pixel transformation is defined as:

  $$x &= 2 \gamma \cos \theta \sin \frac{\phi}{2} \\
    y &= \gamma \sin \theta$$

  where:

  $$\gamma = \frac{180^\circ}{\pi} \sqrt{\frac{2}{1 + \cos \theta \cos(\phi / 2)}}$$

  Invertibility: All ASDF tools are required to provide the inverse of
  this transform.

$ref: "pseudocylindrical-1.0.0"
```

Conic

conic: Base class of all conic projections.

Type: transform-1.0.0 (page 58) and object.

Base class of all conic projections.

In conic projections, the sphere is thought to be projected onto the surface of a cone which is then opened out.

In a general sense, the pixel-to-sky transformation is defined as:

$$\phi = \arg\left(\frac{Y_0 - y}{R_\theta}, \frac{x}{R_\theta}\right) / C$$
$$R_\theta = \mathrm{sign} \theta_a \sqrt{x^2 + (Y_0 - y)^2}$$

and the inverse (sky-to-pixel) is defined as:

$$x = R_\theta \sin(C \phi)$$
$$y = R_\theta \cos(C \phi) + Y_0$$

where C is the "constant of the cone":

$$C = \frac{180^\circ \cos \theta}{\pi R_\theta}$$

All of:

`0`

Type: transform-1.0.0 (page 58).

`1`

Type: object.

Properties:

`direction`

Type: any from [”pix2sky”, “sky2pix”].

Default: “pix2sky”

`sigma`

Type: number.

$(\theta_1 + \theta_2)/2$ where θ_1 and θ_2 are the latitudes of the standard parallels, in degrees.

Default: 0

`delta`

Type: number.

$(\theta_1 - \theta_2)/2$ where θ_1 and θ_2 are the latitudes of the standard parallels, in degrees.

Default: 0

Original schema in YAML

```yaml
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/transform/conic-1.0.0"
title: |
  Base class of all conic projections.

description: |
  In conic projections, the sphere is thought to be projected onto the
  surface of a cone which is then opened out.

  In a general sense, the pixel-to-sky transformation is defined as:

  $$\phi &= \arg\left(\frac{Y_0 - y}{R_\theta}, \frac{x}{R_\theta}\right) / C \\
    R_\theta &= \mathrm{sign} \theta_a \sqrt{x^2 + (Y_0 - y)^2}$$

  and the inverse (sky-to-pixel) is defined as:

  $$x &= R_\theta \sin (C \phi) \\
    y &= R_\theta \cos (C \phi) + Y_0$$

  where $C$ is the "constant of the cone":

  $$C = \frac{180^\circ \cos \theta}{\pi R_\theta}$$

allOf:
  - $ref: "transform-1.0.0"
  - type: object
    properties:
      direction:
        enum: [pix2sky, sky2pix]
        default: pix2sky

      sigma:
        type: number
        description: |
          $(\theta_1 + \theta_2) / 2$ where $\theta_1$ and $\theta_2$
          are the latitudes of the standard parallels, in degrees.
        default: 0

      delta:
        type: number
        description: |
          $(\theta_1 - \theta_2) / 2$ where $\theta_1$ and $\theta_2$
```

```
      are the latitudes of the standard parallels, in degrees.
    default: 0
```

conic_perspective: Colles' conic perspecitve projection.

Type: conic-1.0.0 (page 102).

Colles' conic perspecitve projection.

Corresponds to the `COP` projection in the FITS WCS standard.

See *conic* (page 102) for the definition of the full transformation.

The transformation is defined as:

$$C = \sin \theta_a$$
$$R_\theta = \frac{180^\circ}{\pi} \cos \eta [\cot \theta_a - \tan(\theta - \theta_a)]$$
$$Y_0 = \frac{180^\circ}{\pi} \cos \eta \cot \theta_a$$

Invertibility: All ASDF tools are required to provide the inverse of this transform.

Original schema in YAML


```yaml
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/transform/conic_perspective-1.0.0"
tag: "tag:stsci.edu:asdf/transform/conic_perspective-1.0.0"
title: |
  Colles' conic perspecitve projection.

description: |
  Corresponds to the `COP` projection in the FITS WCS standard.

  See
  [conic](ref:http://stsci.edu/schemas/asdf/transform/conic-1.0.0)
  for the definition of the full transformation.

  The transformation is defined as:

  $$C &= \sin \theta_a \\
    R_\theta &= \frac{180^\circ}{\pi} \cos \eta [ \cot \theta_a - \tan(\theta - \theta_a)] \\
    Y_0 &= \frac{180^\circ}{\pi} \cos \eta \cot \theta_a$$

  Invertibility: All ASDF tools are required to provide the inverse of
  this transform.

$ref: "conic-1.0.0"
```


conic_equidistant: Conic equidistant projection.

Type: conic-1.0.0 (page 102).

Conic equidistant projection.

Corresponds to the COD projection in the FITS WCS standard.

See *conic* (page 102) for the definition of the full transformation.

The transformation is defined as:

$$\begin{aligned} C &= \frac{180^\circ}{\pi} \frac{\sin\theta_a\sin\eta}{\eta} \\ R_\theta &= \theta_a - \theta + \eta\cot\eta\cot\theta_a \\ Y_0 &= \eta\cot\eta\cot\theta_a \end{aligned}$$

Invertibility: All ASDF tools are required to provide the inverse of this transform.

Original schema in YAML

```yaml
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/transform/conic_equidistant-1.0.0"
tag: "tag:stsci.edu:asdf/transform/conic_equidistant-1.0.0"
title: |
  Conic equidistant projection.

description: |
  Corresponds to the `COD` projection in the FITS WCS standard.

  See
  [conic](ref:http://stsci.edu/schemas/asdf/transform/conic-1.0.0)
  for the definition of the full transformation.

  The transformation is defined as:

  $$C &= \frac{180^\circ}{\pi} \frac{\sin\theta_a\sin\eta}{\eta} \\
    R_\theta &= \theta_a - \theta + \eta\cot\eta\cot\theta_a \\
    Y_0 = \eta\cot\eta\cot\theta_a$$

  Invertibility: All ASDF tools are required to provide the inverse of
  this transform.

$ref: "conic-1.0.0"
```

conic_equal_area: Alber's conic equal area projection.

Type: conic-1.0.0 (page 102).

Alber's conic equal area projection.

Corresponds to the COE projection in the FITS WCS standard.

See *conic* (page 102) for the definition of the full transformation.

The transformation is defined as:

$$\begin{aligned} C &= \gamma/2 \\ R_\theta &= \frac{180^\circ}{\pi} \frac{2}{\gamma} \sqrt{1 + \sin\theta_1\sin\theta_2 - \gamma\sin\theta} \\ Y_0 &= \frac{180^\circ}{\pi} \frac{2}{\gamma} \sqrt{1 + \sin\theta_1\sin\theta_2 - \gamma\sin((\theta_1 + \theta_2)/2)} \end{aligned}$$

where:

$$\gamma = \sin\theta_1 + \sin\theta_2$$

Invertibility: All ASDF tools are required to provide the inverse of this transform.

Original schema in YAML

```
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/transform/conic_equal_area-1.0.0"
tag: "tag:stsci.edu:asdf/transform/conic_equal_area-1.0.0"
title: |
  Alber's conic equal area projection.

description: |
  Corresponds to the `COE` projection in the FITS WCS standard.

  See
  [conic](ref:http://stsci.edu/schemas/asdf/transform/conic-1.0.0)
  for the definition of the full transformation.

  The transformation is defined as:

  $$C &= \gamma / 2 \\
    R_\theta &= \frac{180^\circ}{\pi} \frac{2}{\gamma} \sqrt{1 + \sin \theta_1 \sin \theta_2 - \gamma \sin \theta} \\
    Y_0 &= \frac{180^\circ}{\pi} \frac{2}{\gamma} \sqrt{1 + \sin \theta_1 \sin \theta_2 - \gamma \sin((\theta_1 + \theta_

  where:

  $$\gamma = \sin \theta_1 + \sin \theta_2$$

  Invertibility: All ASDF tools are required to provide the inverse of
  this transform.

$ref: "conic-1.0.0"
```

conic_orthomorphic: Conic orthomorphic projection.

Type: conic-1.0.0 (page 102).

Conic orthomorphic projection.

Corresponds to the `COO` projection in the FITS WCS standard.

See *conic* (page 102) for the definition of the full transformation.

The transformation is defined as:

$$C = \frac{\ln\left(\frac{\cos\theta_2}{\cos\theta_1}\right)}{\ln\left[\frac{\tan\left(\frac{90^\circ-\theta_2}{2}\right)}{\tan\left(\frac{90^\circ-\theta_1}{2}\right)}\right]}$$

$$R_\theta = \psi\left[\tan\left(\frac{90^\circ-\theta}{2}\right)\right]^C$$

$$Y_0 = \psi\left[\tan\left(\frac{90^\circ-\theta_a}{2}\right)\right]^C$$

where:

$$\psi = \frac{180^\circ}{\pi} \frac{\cos\theta}{C\left[\tan\left(\frac{90^\circ-\theta}{2}\right)\right]^C}$$

Invertibility: All ASDF tools are required to provide the inverse of this transform.

Original schema in YAML

```
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/transform/conic_orthomorphic-1.0.0"
tag: "tag:stsci.edu:asdf/transform/conic_orthomorphic-1.0.0"
title: |
  Conic orthomorphic projection.

description: |
  Corresponds to the `COO` projection in the FITS WCS standard.

  See
  [conic](ref:http://stsci.edu/schemas/asdf/transform/conic-1.0.0)
  for the definition of the full transformation.

  The transformation is defined as:

  $$C &= \frac{\ln \left( \frac{\cos\theta_2}{\cos\theta_1} \right)}
            {\ln \left[ \frac{\tan\left(\frac{90^\circ-\theta_2}{2}\right)}
                              {\tan\left(\frac{90^\circ-\theta_1}{2}\right)} \right] } \\
    R_\theta &= \psi \left[ \tan \left( \frac{90^\circ - \theta}{2} \right) \right]^C \\
    Y_0 &= \psi \left[ \tan \left( \frac{90^\circ - \theta_a}{2} \right) \right]^C$$

  where:

  $$\psi = \frac{180^\circ}{\pi} \frac{\cos \theta}
           {C\left[\tan\left(\frac{90^\circ-\theta}{2}\right)\right]^C}$$

  Invertibility: All ASDF tools are required to provide the inverse of
  this transform.

$ref: "conic-1.0.0"
```

Pseudoconic

pseudoconic: Base class of all pseudoconic projections.

Type: transform-1.0.0 (page 58) and object.

Base class of all pseudoconic projections.

Pseudoconics are a subclass of conics with concentric parallels.

All of:

`0`

Type: transform-1.0.0 (page 58).

1

Type: object.

Properties:

`direction`

Type: any from [”pix2sky”, “sky2pix”].

Default: “pix2sky”

Original schema in YAML

```
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/transform/pseudoconic-1.0.0"
title: |
  Base class of all pseudoconic projections.

description: |
  Pseudoconics are a subclass of conics with concentric parallels.

allOf:
  - $ref: "transform-1.0.0"
  - type: object
    properties:
      direction:
        enum: [pix2sky, sky2pix]
        default: pix2sky
```

bonne_equal_area: Bonne’s equal area pseudoconic projection.

Type: pseudoconic-1.0.0 (page 107) and object.

Bonne’s equal area pseudoconic projection.

Corresponds to the `BON` projection in the FITS WCS standard.

The pixel-to-sky transformation is defined as:

$$\phi = \frac{\pi}{180^\circ} A_\phi R_\theta / \cos\theta$$
$$\theta = Y_0 - R_\theta$$

where:

$$R_\theta = \mathrm{sign}\theta_1 \sqrt{x^2 + (Y_0 - y)^2}$$
$$A_\phi = \arg\left(\frac{Y_0 - y}{R_\theta}, \frac{x}{R_\theta}\right)$$

And the sky-to-pixel transformation is defined as:

$$x = R_\theta \sin A_\phi$$
$$y = -R_\theta \cos A_\phi + Y_0$$

where:

$$A_\phi = \frac{180^\circ}{\pi R_\theta} \phi \cos \theta$$
$$R_\theta = Y_0 - \theta$$
$$Y_0 = \frac{180^\circ}{\pi} \cot \theta_1 + \theta_1$$

Invertibility: All ASDF tools are required to provide the inverse of this transform.

All of:

`0`

Type: pseudoconic-1.0.0 (page 107).

`1`

Type: object.

Properties:

`theta1`

Type: number.

Bonne conformal latitude, in degrees.

Default: 0

Original schema in YAML

```yaml
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/transform/bonne_equal_area-1.0.0"
tag: "tag:stsci.edu:asdf/transform/bonne_equal_area-1.0.0"
title: |
  Bonne's equal area pseudoconic projection.

description: |
  Corresponds to the `BON` projection in the FITS WCS standard.

  The pixel-to-sky transformation is defined as:

  $$\phi &= \frac{\pi}{180^\circ} A_\phi R_\theta / \cos \theta \\
    \theta &= Y_0 - R_\theta$$

  where:

  $$R_\theta &= \mathrm{sign} \theta_1 \sqrt{x^2 + (Y_0 - y)^2} \\
    A_\phi &= \arg\left(\frac{Y_0 - y}{R_\theta}, \frac{x}{R_\theta}\right)$$

  And the sky-to-pixel transformation is defined as:

  $$x &= R_\theta \sin A_\phi \\
    y &= -R_\theta \cos A_\phi + Y_0$$

  where:

  $$A_\phi &= \frac{180^\circ}{\pi R_\theta} \phi \cos \theta \\
    R_\theta &= Y_0 - \theta \\
```

```
    Y_0 &= \frac{180^\circ}{\pi} \cot \theta_1 + \theta_1$$

  Invertibility: All ASDF tools are required to provide the inverse of
  this transform.

allOf:
  - $ref: "pseudoconic-1.0.0"
  - type: object
    properties:
      theta1:
        type: number
        description: |
          Bonne conformal latitude, in degrees.
        default: 0
```

polyconic: Polyconic projection.

Type: pseudoconic-1.0.0 (page 107).

Polyconic projection.

Corresponds to the PCO projection in the FITS WCS standard.

Invertibility: All ASDF tools are required to provide the inverse of this transform.

Original schema in YAML

```yaml
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/transform/polyconic-1.0.0"
tag: "tag:stsci.edu:asdf/transform/polyconic-1.0.0"
title: |
  Polyconic projection.

description: |
  Corresponds to the `PCO` projection in the FITS WCS standard.

  Invertibility: All ASDF tools are required to provide the inverse of
  this transform.

$ref: "pseudoconic-1.0.0"
```

Quadcube

quadcube: Base class of all quadcube projections.

Type: transform-1.0.0 (page 58) and object.

Base class of all quadcube projections.

Quadrilateralized spherical cube (quad-cube) projections belong to the class of polyhedral projections in which the sphere is projected onto the surface of an enclosing polyhedron.

The six faces of the quad-cube projections are numbered and laid out as:

```
      0
4 3 2 1 4 3 2
      5
```

All of:

`0`

Type: transform-1.0.0 (page 58).

`1`

Type: object.

Properties:

`direction`

Type: any from ["pix2sky", "sky2pix"].

Default: "pix2sky"

Original schema in YAML


```yaml
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/transform/quadcube-1.0.0"
title: |
  Base class of all quadcube projections.

description: |
  Quadrilateralized spherical cube (quad-cube) projections belong to
  the class of polyhedral projections in which the sphere is projected
  onto the surface of an enclosing polyhedron.

  The six faces of the quad-cube projections are numbered and laid out
  as:

  ```
            0
      4 3 2 1 4 3 2
            5
  ```

allOf:
  - $ref: "transform-1.0.0"
  - type: object
    properties:
      direction:
        enum: [pix2sky, sky2pix]
        default: pix2sky
```

tangential_spherical_cube: Tangential spherical cube projection.

Type: quadcube-1.0.0 (page 110).

Tangential spherical cube projection.

Corresponds to the TSC projection in the FITS WCS standard.

Invertibility: All ASDF tools are required to provide the inverse of this transform.

Original schema in YAML

```yaml
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/transform/tangential_spherical_cube-1.0.0"
tag: "tag:stsci.edu:asdf/transform/tangential_spherical_cube-1.0.0"
title: |
  Tangential spherical cube projection.

description: |
  Corresponds to the `TSC` projection in the FITS WCS standard.

  Invertibility: All ASDF tools are required to provide the inverse of
  this transform.

$ref: "quadcube-1.0.0"
```

cobe_quad_spherical_cube: COBE quadrilateralized spherical cube projection.

Type: quadcube-1.0.0 (page 110).

COBE quadrilateralized spherical cube projection.

Corresponds to the CSC projection in the FITS WCS standard.

Invertibility: All ASDF tools are required to provide the inverse of this transform.

Original schema in YAML

```yaml
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/transform/cobe_quad_spherical_cube-1.0.0"
tag: "tag:stsci.edu:asdf/transform/cobe_quad_spherical_cube-1.0.0"
title: |
  COBE quadrilateralized spherical cube projection.

description: |
  Corresponds to the `CSC` projection in the FITS WCS standard.

  Invertibility: All ASDF tools are required to provide the inverse of
  this transform.

$ref: "quadcube-1.0.0"
```

quad_spherical_cube: Quadrilateralized spherical cube projection.

Type: quadcube-1.0.0 (page 110).

Quadrilateralized spherical cube projection.

Corresponds to the QSC projection in the FITS WCS standard.

Invertibility: All ASDF tools are required to provide the inverse of this transform.

Original schema in YAML

```yaml
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/transform/quad_spherical_cube-1.0.0"
tag: "tag:stsci.edu:asdf/transform/quad_spherical_cube-1.0.0"
title: |
  Quadrilateralized spherical cube projection.

description: |
  Corresponds to the `QSC` projection in the FITS WCS standard.

  Invertibility: All ASDF tools are required to provide the inverse of
  this transform.

$ref: "quadcube-1.0.0"
```

HEALPix

healpix: HEALPix projection.

Type: transform-1.0.0 (page 58) and object.

HEALPix projection.

Corresponds to the `XPH` projection in the FITS WCS standard.

Invertibility: All ASDF tools are required to provide the inverse of this transform.

All of:

`0`

Type: transform-1.0.0 (page 58).

`1`

Type: object.

Properties:

`direction`

Type: any from ["pix2sky", "sky2pix"].

Default: "pix2sky"

`H`

Type: number.

The number of facets in the longitude direction.

Default: 4.0

`X`

Type: number.

The number of facets in the latitude direction.

Default: 3.0

Original schema in YAML

```yaml
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/transform/healpix-1.0.0"
tag: "tag:stsci.edu:asdf/transform/healpix-1.0.0"
title: |
  HEALPix projection.

description: |
  Corresponds to the `XPH` projection in the FITS WCS standard.

  Invertibility: All ASDF tools are required to provide the inverse of
  this transform.

allOf:
  - $ref: "transform-1.0.0"
  - type: object
    properties:
      direction:
        enum: [pix2sky, sky2pix]
        default: pix2sky

      H:
        type: number
        description: |
          The number of facets in the longitude direction.
        default: 4.0

      X:
        type: number
        description: |
          The number of facets in the latitude direction.
        default: 3.0
```

healpix_polar: HEALPix polar, aka "butterfly", projection.

Type: transform-1.0.0 (page 58) and object.

HEALPix polar, aka "butterfly", projection.

Corresponds to the `XPH` projection in the FITS WCS standard.

Invertibility: All ASDF tools are required to provide the inverse of this transform.

All of:

`0`

Type: transform-1.0.0 (page 58).

`1`

Type: object.

Properties:

`direction`

Type: any from ["pix2sky", "sky2pix"].

Default: "pix2sky"

Original schema in YAML


```yaml
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/transform/healpix_polar-1.0.0"
tag: "tag:stsci.edu:asdf/transform/healpix_polar-1.0.0"
title: |
  HEALPix polar, aka "butterfly", projection.

description: |
  Corresponds to the `XPH` projection in the FITS WCS standard.

  Invertibility: All ASDF tools are required to provide the inverse of
  this transform.

allOf:
  - $ref: "transform-1.0.0"
  - type: object
    properties:
      direction:
        enum: [pix2sky, sky2pix]
        default: pix2sky
```


5.5.6 Polynomials

polynomial: A Polynomial model.

Type: object.

A Polynomial model.

A polynomial model represented by its coefficients stored in an ndarray of shape $(n+1)$ for univariate polynomials or $(n+1, n+1)$ for polynomials with 2 variables, where n is the highest total degree of the polynomial.

$$P = \sum_{i,j=0}^{i+j=n} c_{ij} * x^i * y^j$$

Invertibility: This transform is not automatically invertible.

Properties:

`coefficients`

Type: ndarray-1.0.0 (page 20) or array. Required.

An array with coefficients.

Any of:

Type: ndarray-1.0.0 (page 20).

Type: array.

Items:

Examples:

$P = 1.2 + 0.3 * x + 56.1 * x^2$:

```
!transform/polynomial-1.0.0
  coefficients: !core/ndarray-1.0.0
                [1.2, 0.3, 56.1]
```

$P = 1.2 + 0.3 * x + 3 * x * y + 2.1 * y^2$:

```
!transform/polynomial-1.0.0
  coefficients: !core/ndarray-1.0.0
                [[1.2, 0.0, 2.1],
                 [0.3, 3.0, 0.0],
                 [0.0, 0.0, 0.0]]
```

Original schema in YAML

```
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/transform/polynomial-1.0.0"
tag: "tag:stsci.edu:asdf/transform/polynomial-1.0.0"
title: >
  A Polynomial model.

description: |
  A polynomial model represented by its coefficients stored in
  an ndarray of shape $(n+1)$ for univariate polynomials or $(n+1, n+1)$
  for polynomials with 2 variables, where $n$ is the highest total degree
  of the polynomial.

  $$P = \sum_{i, j=0}^{i+j=n}c_{ij} * x^{i} * y^{j}$$

  Invertibility: This transform is not automatically invertible.

examples:
  -
    - $P = 1.2 + 0.3 * x + 56.1 * x^{2}$
    - |
        !transform/polynomial-1.0.0
          coefficients: !core/ndarray-1.0.0
                        [1.2, 0.3, 56.1]
  -
    - $P = 1.2 + 0.3 * x + 3 * x * y + 2.1 * y^{2}$
    - |
        !transform/polynomial-1.0.0
          coefficients: !core/ndarray-1.0.0
                        [[1.2, 0.0, 2.1],
                         [0.3, 3.0, 0.0],
                         [0.0, 0.0, 0.0]]

type: object
properties:
  coefficients:
    description: |
      An array with coefficients.
```

```
    anyOf:
      - $ref: "../core/ndarray-1.0.0"
      - type: array
required: [coefficients]
```

5.5.7 Regions and labels

regions_selector: Represents a discontinuous transform.

Type: transform-1.0.0 (page 58) and object.

Represents a discontinuous transform.

Maps regions to transgorms and evaluates the transforms with the corresponding inputs.

All of:

`0`

Type: transform-1.0.0 (page 58).

`1`

Type: object.

Properties:

`label_mapper`

Type: label_mapper-1.0.0 (page 120). Required.

An instance of *label_mapper-1.0.0* (page 120)

`inputs`

Type: array of (string). Required.

Names of inputs.

Items:

Type: string.

`outputs`

Type: array of (string). Required.

Names of outputs.

Items:

Type: string.

`selector`

Type: object. Required.

A mapping of regions to trransforms.

Properties:

`labels`

Type: array of (integer or string).

An array of unique region labels.

Items:

Type: integer or string.

`transforms`

Type: array of (transform-1.0.0 (page 58)).

A transform for each region. The order should match the order of labels.

Items:

Type: transform-1.0.0 (page 58).

`undefined_transform_value`

Type: number.

Value to be returned if there's no transform defined for the inputs.

Examples:

Create a regions_selector schema for 2 regions, labeled "1" and "2".:

```
!transform/regions_selector-1.0.0
  inputs: [x, y]
  label_mapper: !transform/label_mapper-1.0.0
    mapper: !core/ndarray-1.0.0
      datatype: int8
      data:
        [[0, 1, 1, 0, 2, 0],
         [0, 1, 1, 0, 2, 0],
         [0, 1, 1, 0, 2, 0],
         [0, 1, 1, 0, 2, 0],
         [0, 1, 1, 0, 2, 0]]

  outputs: [ra, dec, lam]
  selector:
    1: !transform/compose-1.0.0
      forward:
      - !transform/remap_axes-1.0.0
        mapping: [0, 1, 1]
      - !transform/concatenate-1.0.0
        forward:
        - !transform/concatenate-1.0.0
          forward:
          - !transform/shift-1.0.0 {offset: 1.0}
          - !transform/shift-1.0.0 {offset: 2.0}
        - !transform/shift-1.0.0 {offset: 3.0}
    2: !transform/compose-1.0.0
      forward:
      - !transform/remap_axes-1.0.0
        mapping: [0, 1, 1]
      - !transform/concatenate-1.0.0
        forward:
        - !transform/concatenate-1.0.0
          forward:
          - !transform/scale-1.0.0 {factor: 2.0}
          - !transform/scale-1.0.0 {factor: 3.0}
        - !transform/scale-1.0.0 {factor: 3.0}
  undefined_transform_value: .nan
```

Original schema in YAML

```yaml
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/transform/regions_selector-1.0.0"
tag: "tag:stsci.edu:asdf/transform/regions_selector-1.0.0"
title: >
  Represents a discontinuous transform.
description: |
  Maps regions to transgorms and evaluates the transforms with the corresponding inputs.

examples:
  -
    - Create a regions_selector schema for 2 regions, labeled "1" and "2".
    - |
        !transform/regions_selector-1.0.0
          inputs: [x, y]
          label_mapper: !transform/label_mapper-1.0.0
            mapper: !core/ndarray-1.0.0
              datatype: int8
              data:
                [[0, 1, 1, 0, 2, 0],
                 [0, 1, 1, 0, 2, 0],
                 [0, 1, 1, 0, 2, 0],
                 [0, 1, 1, 0, 2, 0],
                 [0, 1, 1, 0, 2, 0]]

          outputs: [ra, dec, lam]
          selector:
            1: !transform/compose-1.0.0
              forward:
              - !transform/remap_axes-1.0.0
                mapping: [0, 1, 1]
              - !transform/concatenate-1.0.0
                forward:
                - !transform/concatenate-1.0.0
                  forward:
                  - !transform/shift-1.0.0 {offset: 1.0}
                  - !transform/shift-1.0.0 {offset: 2.0}
                - !transform/shift-1.0.0 {offset: 3.0}
            2: !transform/compose-1.0.0
              forward:
              - !transform/remap_axes-1.0.0
                mapping: [0, 1, 1]
              - !transform/concatenate-1.0.0
                forward:
                - !transform/concatenate-1.0.0
                  forward:
                  - !transform/scale-1.0.0 {factor: 2.0}
                  - !transform/scale-1.0.0 {factor: 3.0}
                - !transform/scale-1.0.0 {factor: 3.0}
          undefined_transform_value: .nan


allOf:
  - $ref: "transform-1.0.0"
  - type: object
```

```
  properties:
    label_mapper:
      description: |
        An instance of
        [label_mapper-1.0.0](ref:http://stsci.edu/schemas/asdf/transform/label_mapper-1.0.0)
      $ref: "./label_mapper-1.0.0"
    inputs:
      description: |
        Names of inputs.
      type: array
      items:
        type: string
    outputs:
      description: |
        Names of outputs.
      type: array
      items:
        type: string
    selector:
      description: |
        A mapping of regions to trransforms.
      type: object
      properties:
        labels:
          description: |
            An array of unique region labels.
          type: array
          items:
            type:
              - integer
              - string
        transforms:
          description: |
            A transform for each region. The order should match the order of labels.
          type: array
          items:
            $ref: "transform-1.0.0"
    undefined_transform_value:
      description: |
        Value to be returned if there's no transform defined for the inputs.
      type: number
  required: [label_mapper, inputs, outputs, selector]
```

label_mapper: Represents a mapping from a coordinate value to a label.

Type: transform-1.0.0 (page 58) and object.

Represents a mapping from a coordinate value to a label.

A label mapper instance maps inputs to a label. It is used together with *regions_selector* (page 117). The *label_mapper* (page 120) returns the label corresponding to given inputs. The *regions_selector* (page 117) returns the transform corresponding to this label. This maps inputs (e.g. pixels on a detector) to transforms uniquely.

All of:

`0`

Type: transform-1.0.0 (page 58).

1

Type: object.

Properties:

`mapper`

Type: ndarray-1.0.0 (page 20) or object. Required.

An array with the shape of the detector/observation. Pixel values are of type integer or string and represent region labels. Pixels which are not within any region have value 0 or " ".

Any of:

Type: ndarray-1.0.0 (page 20).

Type: object.

Properties:

`labels`

Type: array of (number or array of (number)).

Items:

Type: number or array of (number).

Any of:

Type: number.

Type: array of (number).

Items:

Type: number.

`models`

Type: array of (transform-1.0.0 (page 58)).

Items:

Type: transform-1.0.0 (page 58).

`inputs`

Type: array of (string).

Items:

Type: string.

`inputs_mapping`

Type: transform-1.0.0 (page 58).

Examples:

Map array indices are to labels.:

```
!transform/label_mapper-1.0.0
  mapper: !core/ndarray-1.0.0
    [[1, 0, 2],
    [1, 0, 2],
    [1, 0, 2]]
```

Map numbers dictionary to transforms which return labels.:

```
!transform/label_mapper-1.0.0
  mapper: !!omap
  - !!omap
    labels: [-1.67833272, -1.9580548, -1.118888]
  - !!omap
    models:
  - !transform/compose-1.0.0
    forward:
    - !transform/remap_axes-1.0.0
      mapping: [1]
    - !transform/shift-1.0.0 {offset: 6.0}
  - !transform/compose-1.0.0
    forward:
    - !transform/remap_axes-1.0.0
      mapping: [1]
    - !transform/shift-1.0.0 {offset: 2.0}
  - !transform/compose-1.0.0
    forward:
    - !transform/remap_axes-1.0.0
      mapping: [1]
    - !transform/shift-1.0.0 {offset: 4.0}
  inputs: [x, y]
  inputs_mapping: !transform/remap_axes-1.0.0
    mapping: [0]
    n_inputs: 2
```

Map a number wihtin a range of numbers to transforms which return labels.:

```
!transform/label_mapper-1.0.0
  mapper: !!omap
  - !!omap
    labels:
    - [3.2, 4.1]
    - [2.67, 2.98]
    - [1.95, 2.3]
  - !!omap
    models:
  - !transform/compose-1.0.0
    forward:
    - !transform/remap_axes-1.0.0
      mapping: [1]
    - !transform/shift-1.0.0 {offset: 6.0}
  - !transform/compose-1.0.0
    forward:
    - !transform/remap_axes-1.0.0
      mapping: [1]
    - !transform/shift-1.0.0 {offset: 2.0}
  - !transform/compose-1.0.0
    forward:
    - !transform/remap_axes-1.0.0
```

```
      mapping: [1]
    - !transform/shift-1.0.0 {offset: 4.0}
  inputs: [x, y]
  inputs_mapping: !transform/remap_axes-1.0.0
    mapping: [0]
    n_inputs: 2
```

Original schema in YAML

```
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/transform/label_mapper-1.0.0"
tag: "tag:stsci.edu:asdf/transform/label_mapper-1.0.0"
title: >
  Represents a mapping from a coordinate value to a label.
description: |
  A label mapper instance maps inputs to a label.  It is used together
  with
  [regions_selector](ref:http://stsci.edu/schemas/asdf/transform/regions_selector-1.0.0). The
  [label_mapper](ref:http://stsci.edu/schemas/asdf/transform/label_mapper-1.0.0)
  returns the label corresponding to given inputs. The
  [regions_selector](ref:http://stsci.edu/schemas/asdf/transform/regions_selector-1.0.0)
  returns the transform corresponding to this label. This maps inputs
  (e.g. pixels on a detector) to transforms uniquely.

examples:
  -
    - Map array indices are to labels.

    - |
        !transform/label_mapper-1.0.0
          mapper: !core/ndarray-1.0.0
            [[1, 0, 2],
            [1, 0, 2],
            [1, 0, 2]]

  -
    - Map numbers dictionary to transforms which return labels.

    - |
        !transform/label_mapper-1.0.0
          mapper: !!omap
          - !!omap
            labels: [-1.67833272, -1.9580548, -1.118888]
          - !!omap
            models:
          - !transform/compose-1.0.0
            forward:
            - !transform/remap_axes-1.0.0
              mapping: [1]
            - !transform/shift-1.0.0 {offset: 6.0}
          - !transform/compose-1.0.0
            forward:
            - !transform/remap_axes-1.0.0
              mapping: [1]
```

```
            - !transform/shift-1.0.0 {offset: 2.0}
          - !transform/compose-1.0.0
            forward:
            - !transform/remap_axes-1.0.0
              mapping: [1]
            - !transform/shift-1.0.0 {offset: 4.0}
          inputs: [x, y]
          inputs_mapping: !transform/remap_axes-1.0.0
            mapping: [0]
            n_inputs: 2

  -
    - Map a number wihtin a range of numbers to transforms which return labels.

    - |
        !transform/label_mapper-1.0.0
          mapper: !!omap
          - !!omap
            labels:
            - [3.2, 4.1]
            - [2.67, 2.98]
            - [1.95, 2.3]
          - !!omap
            models:
          - !transform/compose-1.0.0
            forward:
            - !transform/remap_axes-1.0.0
              mapping: [1]
            - !transform/shift-1.0.0 {offset: 6.0}
          - !transform/compose-1.0.0
            forward:
            - !transform/remap_axes-1.0.0
              mapping: [1]
            - !transform/shift-1.0.0 {offset: 2.0}
          - !transform/compose-1.0.0
            forward:
            - !transform/remap_axes-1.0.0
              mapping: [1]
            - !transform/shift-1.0.0 {offset: 4.0}
          inputs: [x, y]
          inputs_mapping: !transform/remap_axes-1.0.0
            mapping: [0]
            n_inputs: 2

allOf:
  - $ref: "transform-1.0.0"
  - type: object
    properties:
      mapper:
        description: |
          An array with the shape of the detector/observation.
          Pixel values are of type integer or string and represent
          region labels.
          Pixels which are not within any region have value 0 or " ".
        anyOf:
          - $ref: "../core/ndarray-1.0.0"
          - type: object
            properties:
```

```
              labels:
                type: array
                items:
                  anyOf:
                    - type: number
                    - type: array
                      items:
                        type: number
                      minLength: 2
                      maxLength: 2
              models:
                type: array
                items:
                  $ref: "transform-1.0.0"

      inputs:
        type: array
        items:
          type: string
      inputs_mapping:
        $ref: "transform-1.0.0"

    required: [mapper]
```

5.6 WCS

The WCS module contains schema used to describe generalized world coordinate system transformations.

Requires:

Core (page 17), Unit (page 46), Transform (page 58)

5.6.1 wcs: A system for describing generalized world coordinate transformations.

Type: object.

A system for describing generalized world coordinate transformations.

ASDF WCS is a way of specifying transformations (usually from detector space to world coordinate space and back) by using the transformations in the `transform-schema` module.

Properties:

`name`

Type: string. Required.

A descriptive name for this WCS.

`steps`

Type: array of (step-1.0.0 (page 126)). Required.

A list of steps in the forward transformation from detector to world coordinates. The inverse transformation is determined automatically by reversing this list, and inverting each of the individual transforms according to the rules described in *inverse* (page 59).

Items:

Type: step-1.0.0 (page 126).

Original schema in YAML

```yaml
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/wcs/wcs-1.0.0"
tag: "tag:stsci.edu:asdf/wcs/wcs-1.0.0"
title: >
  A system for describing generalized world coordinate transformations.
description: >
  ASDF WCS is a way of specifying transformations (usually from
  detector space to world coordinate space and back) by using the
  transformations in the `transform-schema` module.
type: object
properties:
  name:
    description: |
      A descriptive name for this WCS.
    type: string

  steps:
    description: |
      A list of steps in the forward transformation from detector to
      world coordinates.
      The inverse transformation is determined automatically by
      reversing this list, and inverting each of the individual
      transforms according to the rules described in
      [inverse](ref:http://stsci.edu/schemas/asdf/transform/transform-1.0.0/properties/inverse).
    type: array
    items:
      $ref: step-1.0.0

required: [name, steps]
additionalProperties: true
```

5.6.2 step: Describes a single step of a WCS transform pipeline.

Type: object.

Describes a single step of a WCS transform pipeline.

Properties:

`frame`

Type: string or frame-1.0.0 (page 127). Required.

The frame of the inputs to the transform.

Any of:

Type: string.

Type: frame-1.0.0 (page 127).

`transform`

Type: transform-1.0.0 (page 58) or null.

The transform from this step to the next one. The last step in a WCS should not have a transform, but exists only to describe the frames and units of the final output axes.

Default: null

Any of:

Type: transform-1.0.0 (page 58).

Type: null.

Original schema in YAML

```yaml
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/wcs/step-1.0.0"
tag: "tag:stsci.edu:asdf/wcs/step-1.0.0"
title: >
  Describes a single step of a WCS transform pipeline.
description: >
examples: []

type: object
properties:
  frame:
    description: |
      The frame of the inputs to the transform.
    anyOf:
      - type: string
      - $ref: frame-1.0.0

  transform:
    description: |
      The transform from this step to the next one.  The
      last step in a WCS should not have a transform, but
      exists only to describe the frames and units of the
      final output axes.
    anyOf:
      - $ref: ../transform/transform-1.0.0
      - type: 'null'
    default: null

required: [frame]
```

5.6.3 frame: The base class of all coordinate frames.

Type: object.

The base class of all coordinate frames.

These objects are designed to be nested in arbitrary ways to build up transformation pipelines out of a number of low-level pieces.

Most of these coordinate frames are defined in IERS conventions (http://www.iers.org/IERS/EN/Publications/TechnicalNotes/tn36.ht

Properties:

`name`

Type: string. Required.

A user-friendly name for the frame.

`axes_order`

Type: array of (integer).

The order of the axes.

Items:

Type: integer.

`axes_names`

Type: array of (string or null).

The name of each axis in this frame.

Items:

Type: string or null.

Any of:

Type: string.

Type: null.

`reference_frame`

Type: object.

The reference frame.

Properties:

`type`

Type: any from [”ICRS”, “FK5”, “FK4”, “FK4_noeterms”, “galactic”, “galactocentric”, “GCRS”, “CIRS”, “ITRS”, “precessed_geocentric”]. Required.

The reference frame type. Some reference frame types require additional properties, listed next to each reference frame type below.

The reference frames types are:

- `ICRS`
- `FK5`: `equinox`.
- `FK4`: `equinox` and optionally `obstime`.
- `FK4_noeterms`: `equinox` and optionally `obstime`.

- galactic
- galactocentric: galcen_distance, galcen_ra, galcen_dec, z_sun and roll.
- GCRS: obstime, obsgeoloc, and obsgeovel.
- CIRS: obstime.
- ITRS: obstime.
- precessed_geocentric: obstime, obsgeoloc, and obsgeovel.

Default: "ICRS"

equinox

Type: time-1.0.0 (page 48).

The equinox of the reference frame. Required when reference_frame one of:

FK5, FK4, FK4_noeterms

obstime

Type: time-1.0.0 (page 48).

The observation time of the reference frame, used to determine the location of the Earth. Required when reference_frame is one of:

FK4, FK4_noeterms, GCRS, CIRS, ITRS

If not provided, it defaults to the same value as equinox.

galcen_distance

Type: array.

The distance from the Sun to the Galactic center. Required when reference_frame is galactocentric.

Items:

index[0]

Type: number.

index[1]

Type: unit-1.0.0 (page 46).

Default: "pc"

galcen_ra

Type: array.

The Right Ascension (RA) of the Galactic center in the ICRS frame. Required when reference_frame is galactocentric.

Items:

index[0]

Type: number.

index[1]

Type: unit-1.0.0 (page 46).

Default: "deg"

galcen_dec

Type: array.

The Declination (DEC) of the Galactic center in the ICRS frame. Required when reference_frame is galactocentric.

Items:

index[0]

Type: number.

index[1]

Type: unit-1.0.0 (page 46).

Default: "deg"

z_sun

Type: array.

The distance from the sun to the galactic midplane. Required when reference_frame is galactocentric. Required when reference_frame is galactocentric.

Items:

index[0]

Type: number.

index[1]

Type: unit-1.0.0 (page 46).

Default: "pc"

roll

Type: array.

The angle to rotate about the final x-axis, relative to the orientation for galactic. Required when reference_frame is galactocentric.

Items:

index[0]

Type: number.

index[1]

Type: unit-1.0.0 (page 46).

Default: "deg"

obsgeoloc

Type: array.

3-vector giving the position of the observer relative to the center-of-mass of the Earth, oriented the same as BCRS/ICRS. Defaults to [0, 0, 0], meaning "true" GCRS. Used when reference_frame is GCRS or precessed_geocentric.

Default: [[0, 0, 0]]

Items:

index[0]

Type: array of (number) *len* = 3.

Items:

Type: number.

index[1]

Type: unit-1.0.0 (page 46).

Default: "m"

obsgeovel

Type: array.

3-vector giving the velocity of the observer relative to the center-of-mass of the Earth, oriented the same as BCRS/ICRS. Defaults to `[0, 0, 0]`, meaning "true" GCRS. Used when `reference_frame` is `GCRS` or `precessed_geocentric`.

Default: [[0, 0, 0]]

Items:

index[0]

Type: array of (number) *len* = 3.

Items:

Type: number.

index[1]

Type: unit-1.0.0 (page 46).

Default: "m/s"

unit

Type: array of (unit-1.0.0 (page 46)).

Units for each axis.

Items:

Type: unit-1.0.0 (page 46).

Examples:

A celestial frame in the FK4 reference frame.

```
!wcs/celestial_frame-1.0.0
  axes_names: [ra, dec]
  name: CelestialFrame
  reference_frame:
    type: FK4
    equinox: !time/time-1.0.0 '2010-01-01 00:00:00.000'
    obstime: !time/time-1.0.0 '2015-01-01 00:00:00.000'
  unit: [!unit/unit-1.0.0 deg, !unit/unit-1.0.0 deg]
```

Original schema in YAML

```
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/wcs/frame-1.0.0"
title: |
  The base class of all coordinate frames.

description: |
  These objects are designed to be nested in arbitrary ways to build up
  transformation pipelines out of a number of low-level pieces.

  Most of these coordinate frames are defined in [IERS
  conventions](http://www.iers.org/IERS/EN/Publications/TechnicalNotes/tn36.html).

examples:
  -
    - |
        A celestial frame in the FK4 reference frame.
    - |
        !wcs/celestial_frame-1.0.0
          axes_names: [ra, dec]
          name: CelestialFrame
          reference_frame:
            type: FK4
            equinox: !time/time-1.0.0 '2010-01-01 00:00:00.000'
            obstime: !time/time-1.0.0 '2015-01-01 00:00:00.000'
          unit: [!unit/unit-1.0.0 deg, !unit/unit-1.0.0 deg]

type: object
properties:
  name:
    description: |
      A user-friendly name for the frame.
    type: string

  axes_order:
    description: |
      The order of the axes.
    type: array
    items:
      type: integer

  axes_names:
    description: |
      The name of each axis in this frame.
    type: array
    items:
      anyOf:
        - type: string
        - type: 'null'

  reference_frame:
    description: |
      The reference frame.
    type: object
    properties:
```

```
    type:
      description: |
        The reference frame type.  Some reference frame types
        require additional properties, listed next to each reference
        frame type below.

        The reference frames types are:

        - `ICRS`

        - `FK5`: `equinox`.

        - `FK4`: `equinox` and optionally `obstime`.

        - `FK4_noeterms`: `equinox` and optionally `obstime`.

        - `galactic`

        - `galactocentric`: `galcen_distance`, `galcen_ra`,
          `galcen_dec`, `z_sun` and `roll`.

        - `GCRS`: `obstime`, `obsgeoloc`, and `obsgeovel`.

        - `CIRS`: `obstime`.

        - `ITRS`: `obstime`.

        - `precessed_geocentric`: `obstime`, `obsgeoloc`, and
          `obsgeovel`.

      enum: [ICRS, FK5, FK4, FK4_noeterms, galactic, galactocentric,
             GCRS, CIRS, ITRS, precessed_geocentric]
      default: ICRS

    equinox:
      description: |
        The equinox of the reference frame.  Required when
        `reference_frame` one of:

          `FK5`, `FK4`, `FK4_noeterms`

      $ref: ../time/time-1.0.0

    obstime:
      description: |
        The observation time of the reference frame, used to determine
        the location of the Earth.  Required when `reference_frame` is
        one of:

          `FK4`, `FK4_noeterms`, `GCRS`, `CIRS`, `ITRS`

        If not provided, it defaults to the same value as `equinox`.
      $ref: ../time/time-1.0.0

    galcen_distance:
      description: |
        The distance from the Sun to the Galactic center.  Required when
        `reference_frame` is `galactocentric`.
```

```
        type: array
        items:
          - type: number
          - $ref: ../unit/unit-1.0.0
            default: pc

      galcen_ra:
        description: |
          The Right Ascension (RA) of the Galactic center in the ICRS
          frame.  Required when `reference_frame` is `galactocentric`.
        type: array
        items:
          - type: number
          - $ref: ../unit/unit-1.0.0
            default: deg

      galcen_dec:
        description: |
          The Declination (DEC) of the Galactic center in the ICRS frame.
          Required when `reference_frame` is `galactocentric`.
        type: array
        items:
          - type: number
          - $ref: ../unit/unit-1.0.0
            default: deg

      z_sun:
        description: |
          The distance from the sun to the galactic midplane.  Required
          when `reference_frame` is `galactocentric`.  Required when
          `reference_frame` is `galactocentric`.
        type: array
        items:
          - type: number
          - $ref: ../unit/unit-1.0.0
            default: pc

      roll:
        description: |
          The angle to rotate about the final x-axis, relative to the
          orientation for `galactic`.  Required when `reference_frame` is
          `galactocentric`.
        type: array
        items:
          - type: number
          - $ref: ../unit/unit-1.0.0
            default: deg

      obsgeoloc:
        description: |
          3-vector giving the position of the observer relative to the
          center-of-mass of the Earth, oriented the same as
          BCRS/ICRS. Defaults to `[0, 0, 0]`, meaning "true" GCRS.  Used
          when `reference_frame` is `GCRS` or `precessed_geocentric`.
        type: array
        items:
          - type: array
            items:
```

```
              type: number
            minItems: 3
            maxItems: 3
          - $ref: ../unit/unit-1.0.0
            default: m
        default:
          - [0, 0, 0]

      obsgeovel:
        description: |
          3-vector giving the velocity of the observer relative to the
          center-of-mass of the Earth, oriented the same as
          BCRS/ICRS. Defaults to `[0, 0, 0]`, meaning "true" GCRS.  Used
          when `reference_frame` is `GCRS` or `precessed_geocentric`.
        type: array
        items:
          - type: array
            items:
              type: number
            minItems: 3
            maxItems: 3
          - $ref: ../unit/unit-1.0.0
            default: m/s
        default:
          - [0, 0, 0]

    required: [type]

  unit:
    description: |
      Units for each axis.
    type: array
    items:
      $ref: ../unit/unit-1.0.0

required: [name]
additionalProperties: true
```

5.6.4 celestial_frame: Represents a celestial frame.

Type: object and frame-1.0.0 (page 127).

Represents a celestial frame.

All of:

`0`

Type: object.

Properties:

`axes_names`

Type: any.

`axes_order`

Type: any.

`unit`

Type: any.

1

Type: frame-1.0.0 (page 127).

Original schema in YAML

```yaml
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/wcs/celestial_frame-1.0.0"
tag: "tag:stsci.edu:asdf/wcs/celestial_frame-1.0.0"

title: >
  Represents a celestial frame.

allOf:
  - type: object
    properties:
      axes_names:
        minItems: 2
        maxItems: 3

      axes_order:
        minItems: 2
        maxItems: 3

      unit:
        minItems: 2
        maxItems: 3

  - $ref: frame-1.0.0
```

5.6.5 spectral_frame: Represents a spectral frame.

Type: object and frame-1.0.0 (page 127).

Represents a spectral frame.

All of:

0

Type: object.

Properties:

`reference_position`

Type: any from [”geocenter”, “barycenter”, “heliocenter”].

The position of the reference frame.

Default: “geocenter”

`axes_names`

Type: any.

axes_order

Type: any.

unit

Type: any.

1

Type: frame-1.0.0 (page 127).

Original schema in YAML

```
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/wcs/spectral_frame-1.0.0"
tag: "tag:stsci.edu:asdf/wcs/spectral_frame-1.0.0"

title: >
  Represents a spectral frame.
allOf:
  - type: object
    properties:
      reference_position:
        description: |
          The position of the reference frame.
        enum: [geocenter, barycenter, heliocenter]
        default: geocenter

      axes_names:
        minItems: 1
        maxItems: 1

      axes_order:
        minItems: 1
        maxItems: 1

      unit:
        minItems: 1
        maxItems: 1

  - $ref: frame-1.0.0
```

5.6.6 composite_frame: Represents a set of frames.

Type: object and frame-1.0.0 (page 127).

Represents a set of frames.

All of:

0

Type: object.

Properties:

`name`

Type: string.

Name of composite frame.

`frames`

Type: array.

List of frames in the composite frame.

Items:

1

Type: frame-1.0.0 (page 127).

Original schema in YAML

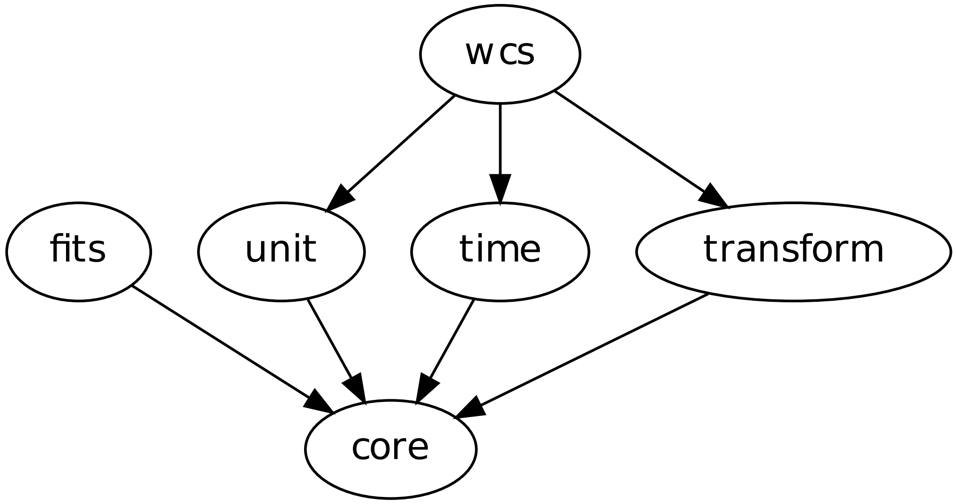
```yaml
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
id: "http://stsci.edu/schemas/asdf/wcs/composite_frame-1.0.0"
tag: "tag:stsci.edu:asdf/wcs/composite_frame-1.0.0"

title: >
  Represents a set of frames.
allOf:
  - type: object
    properties:
      name:
        description:
          Name of composite frame.
        type: string

      frames:
        description:
          List of frames in the composite frame.
        type: array

  - $ref: frame-1.0.0
```

The following graph shows the dependencies between modules:


wcs
fits
unit
time
transform
core

CHAPTER 6

EXTENDING ASDF

ASDF is designed to be extensible so outside teams can add their own types and structures while retaining compatibility with tools that don't understand those conventions.

6.1 YAML Schema

6.1.1 draft: YAML Schema

Type: schema (http://json-schema.org/draft-04/schema) and object.

YAML Schema

A metaschema extending JSON Schema's metaschema to add support for some YAML-specific constructions.

All of:

`0`

Type: schema (http://json-schema.org/draft-04/schema).

`1`

Type: object.

Properties:

`tag`

Type: string ($len \geq 6$).

A fully-qualified YAML tag name that should be associated with the object type returned by the YAML parser; for example, the object must be an instance of the class registered with the parser to create instances of objects with this tag. Implementation of this validator is optional and depends on details of the YAML parser.

`propertyOrder`

Type: array of (string).

Specifies the default order of the properties when writing out. Any keys not listed in propertyOrder will be in arbitrary order at the end.

Items:

Type: string.

`flowStyle`

Type: string from [”block”, “flow”].

Specifies the default serialization style to use for an array or object. YAML supports multiple styles for arrays/sequences and objects/maps, called "block style" and "flow style". For example:

```
Block style: !!map
  Clark : Evans
  Ingy  : döt Net
  Oren  : Ben-Kiki

Flow style: !!map { Clark: Evans, Ingy: döt Net, Oren: Ben-Kiki }
```

This property gives a hint to the tool outputting the YAML which style to use. If not provided, the library is free to use whatever heuristics it wishes to determine the output style. This property does not enforce any particular style on YAML being parsed.

`style`

Type: string from [”inline”, "literal", "folded"].

Specifies the default serialization style to use for a string. YAML supports multiple styles for strings:

```
Inline style: "First line\nSecond line"

Literal style: |
  First line
  Second line

Folded style: >
  First
  line
```

```
Second
line
```

This property gives a hint to the tool outputting the YAML which style to use. If not provided, the library is free to use whatever heuristics it wishes to determine the output style. This property does not enforce any particular style on YAML being parsed.

`examples`

Type: array of (array).

A list of examples to help document the schema. Each pair is a prose description followed by a string containing YAML content.

Items:

Type: array.

Items:

`index[0]`

Type: string.

`index[1]`

Type: string.

Original schema in YAML


```yaml
%YAML 1.1
---
$schema: "http://json-schema.org/draft-04/schema"
id: "http://stsci.edu/schemas/yaml-schema/draft-01"
title:
  YAML Schema
description: |
  A metaschema extending JSON Schema's metaschema to add support for
  some YAML-specific constructions.
allOf:
  - $ref: "http://json-schema.org/draft-04/schema"
  - type: object
    properties:
      tag:
        description: |
          A fully-qualified YAML tag name that should be associated
          with the object type returned by the YAML parser; for
          example, the object must be an instance of the class
          registered with the parser to create instances of objects
          with this tag. Implementation of this validator is optional
          and depends on details of the YAML parser.
        type: string
        minLength: 6

      propertyOrder:
        description: |
          Specifies the default order of the properties when writing
          out.  Any keys not listed in propertyOrder will be in
          arbitrary order at the end.
        type: array
        items:
          type: string

      flowStyle:
        description: |
          Specifies the default serialization style to use for an
          array or object.  YAML supports multiple styles for
          arrays/sequences and objects/maps, called "block style" and
          "flow style".  For example::

            Block style: !!map
              Clark : Evans
              Ingy  : döt Net
              Oren  : Ben-Kiki

            Flow style: !!map { Clark: Evans, Ingy: döt Net, Oren: Ben-Kiki }

          This property gives a hint to the tool outputting the YAML
          which style to use.  If not provided, the library is free to
          use whatever heuristics it wishes to determine the output
          style.  This property does not enforce any particular style
          on YAML being parsed.
        type: string
        enum: [block, flow]

      style:
```

```
        description: |
          Specifies the default serialization style to use for a string.
          YAML supports multiple styles for strings::

            Inline style: "First line\nSecond line"

            Literal style: |
              First line
              Second line

            Folded style: >
              First
              line

              Second
              line

          This property gives a hint to the tool outputting the YAML
          which style to use.  If not provided, the library is free to
          use whatever heuristics it wishes to determine the output
          style.  This property does not enforce any particular style
          on YAML being parsed.
        type: string
        enum: [inline, literal, folded]

      examples:
        description: |
          A list of examples to help document the schema.  Each pair
          is a prose description followed by a string containing YAML
          content.
        type: array
        items:
          type: array
          items:
            - type: string
            - type: string
...
```

YAML Schema (page 141) is a small extension to JSON Schema Draft 4 (http://json-schema.org/latest/json-schema-validation.html) that adds some features specific to YAML.. Understanding JSON Schema (http://spacetelescope.github.io/understanding-json-schema/) provides a good resource for understanding how to use JSON Schema, and further resources are available at json-schema.org (http://json-schema.org). A working understanding of JSON Schema is assumed for this section, which only describes what makes YAML Schema different from JSON Schema.

Writing a new schema is described in *Designing a new tag and schema* (page 148).

6.1.2 `tag` keyword

`tag`, which may be attached to any data type, declares that the element must have the given YAML tag.

For example, the root *asdf* (page 17) schema declares that the `data` property must be an *ndarray* (page 20). It does this not by using the `tag` keyword directly, but by referencing the `ndarray` schema, which in turn has the `tag` keyword. The ASDF schema includes:

```
properties:
  data:
    $ref: "ndarray"
```

And the *ndarray* (page 20) schema includes:

```
tag: "tag:stsci.edu:asdf/core/ndarray-1.0.0"
```

This has the net effect of requiring that the `data` property at the top-level of all ASDF files is tagged as `tag:stsci.edu:asdf/core/ndarray-1.0.0`.

6.1.3 `propertyOrder` keyword

`propertyOrder`, which applies only to objects, declares that the object must have its properties presented in the given order.

TBD: It is not yet clear whether this keyword is necessary or desirable.

6.1.4 `flowStyle` keyword

Must be either `block` or `flow`.

Specifies the default serialization style to use for an array or object. YAML supports multiple styles for arrays/sequences and objects/maps, called "block style" and "flow style". For example:

```
Block style: !!map
 Clark : Evans
 Ingy  : döt Net
 Oren  : Ben-Kiki

Flow style: !!map { Clark: Evans, Ingy: döt Net, Oren: Ben-Kiki }
```

This property gives an optional hint to the tool outputting the YAML which style to use. If not provided, the library is free to use whatever heuristics it wishes to determine the output style. This property does not enforce any particular style on YAML being parsed.

6.1.5 `style` keyword

Must be `inline`, `literal` or `folded`.

Specifies the default serialization style to use for a string. YAML supports multiple styles for strings:

```
Inline style: "First line\nSecond line"

Literal style: |
  First line
  Second line

Folded style: >
  First
  line

  Second
  line
```

This property gives an optional hint to the tool outputting the YAML which style to use. If not provided, the library is free to use whatever heuristics it wishes to determine the output style. This property does not enforce any particular style on YAML being parsed.

6.1.6 `examples` keyword

The schema may contain a list of examples demonstrating how to use the schema. It is a list where each item is a pair. The first item in the pair is a prose description of the example, and the second item is YAML content (as a string) containing the example.

For example:

```
examples:
  -
    - Complex number: 1 real, -1 imaginary
    - "!complex 1-1j"
```

6.2 ASDF Schema

6.2.1 asdf-schema: ASDF schema

Type: draft-01 (http://stsci.edu/schemas/yaml-schema/draft-01) and object.

ASDF schema

A metaschema extending YAML Schema and JSON Schema to add support for some ASDF-specific checks, related to nd-arrays.

All of:

`0`

Type: draft-01 (http://stsci.edu/schemas/yaml-schema/draft-01).

`1`

Type: object.

Properties:

`max_ndim`

Type: integer ≥ 0.

Specifies that the corresponding ndarray is at most the given number of dimensions. If the array has fewer dimensions, it should be logically treated as if it were "broadcast" to the expected dimensions by adding 1's to the front of the shape list.

`ndim`

Type: integer ≥ 0.

Specifies that the matching ndarray is exactly the given number of dimensions.

`datatype`

Type: datatype-1.0.0 (http://stsci.edu/schemas/asdf/core/ndarray-1.0.0#definitions/datatype-1.0.0).

Specifies the datatype of the ndarray.

By default, an array is considered "matching" if the array can be cast to the given datatype without data loss. For exact datatype matching, set `exact_datatype` to `true`.

`exact_datatype`

Type: boolean.

If `true`, the datatype must match exactly.

Default: false

Original schema in YAML

```yaml
%YAML 1.1
---
$schema: "http://json-schema.org/draft-04/schema"
id: "http://stsci.edu/schemas/asdf/asdf-schema-1.0.0"
title:
  ASDF schema
description: |
  A metaschema extending YAML Schema and JSON Schema to add support
  for some ASDF-specific checks, related to nd-arrays.
allOf:
  - $ref: "http://stsci.edu/schemas/yaml-schema/draft-01"
  - type: object
    properties:
      max_ndim:
        description: |
          Specifies that the corresponding ndarray is at most the
          given number of dimensions.  If the array has fewer
          dimensions, it should be logically treated as if it were
          "broadcast" to the expected dimensions by adding 1's to the
          front of the shape list.
        type: integer
        minimum: 0

      ndim:
        description: |
          Specifies that the matching ndarray is exactly the given
          number of dimensions.
        type: integer
        minimum: 0

      datatype:
        description: |
          Specifies the datatype of the ndarray.

          By default, an array is considered "matching" if the array
          can be cast to the given datatype without data loss.  For
          exact datatype matching, set `exact_datatype` to `true`.
        allOf:
          - $ref: "http://stsci.edu/schemas/asdf/core/ndarray-1.0.0#definitions/datatype-1.0.0"

      exact_datatype:
        description: |
          If `true`, the datatype must match exactly.
        type: boolean
        default: false
```

```
...
```

ASDF Schema (page 146) further extends YAML schema to add some validations specific to ASDF, notably to do with *ndarray* (page 20).

6.2.2 `ndim` keyword

Specifies that the matching ndarray is exactly the given number of dimensions.

6.2.3 `max_ndim` keyword

Specifies that the corresponding ndarray is at most the given number of dimensions. If the array has fewer dimensions, it should be logically treated as if it were "broadcast" to the expected dimensions by adding 1's to the front of the shape list.

6.2.4 `datatype` keyword

Specifies the datatype of the ndarray.

By default, an array is considered "matching" if the array can be cast to the given datatype without data loss. For exact datatype matching, set `exact_datatype` to `true`.

6.2.5 `exact_datatype` keyword

If `true`, the datatype must match exactly, rather than just being castable to the given datatype without data loss.

6.3 Designing a new tag and schema

The schema included in the ASDF standard will not be adequate for all needs, but it is possible to mix them with custom schema designed for a specific purpose. It is also possible to extend and specialize an existing schema (described in *Extending an existing schema* (page 151)).

This section will walk through the development of a new tag and schema. In the example, suppose we work at the Space Telescope Science Institute, which can be found on the world wide web at `stsci.edu`. We're developing a new instrument, `FOO`, and we need a way to define the specialized metadata to describe the exposures that it will be generating.

6.3.1 Header

Every ASDF schema should begin with the following header:

```
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
```

This declares that the file is `YAML 1.1` format, and that the structure of the content conforms to `YAML Schema` defined above.

6.3.2 Tags and IDs

All of the tags defined by the ASDF standard itself have the following prefix:

```
tag:stsci.edu:asdf/
```

This prefix is reserved for tags and schemas defined within the ASDF standard itself. ASDF can, of course, include any tags, as long as the tag names are globally unique. So, for our example instrument, we'll declare the tag to be:

```
tag:stsci.edu:FOO/metadata-1.0.0
```

Each tag should be associated with a schema in order to validate it. Each schema must also have a universally unique `id`, which is in the form of unique URI. For the ASDF built-in tags, the mapping from tag name to schema URI is quite simple:

```
tag:stsci.edu:XXX
```

maps to:

```
http://stsci.edu/schemas/XXX
```

Note that this URI doesn't actually have to resolve to anything. In fact, visiting that URL in your web browser is likely to bring up a `404` error. All that's necessary is that it is universally unique and that the tool reading the ASDF file is able to map from a tag name to a schema URI, and then load the associated schema.

Again following with our example, we will assign the following URI to refer to our schema:

```
http://stsci.edu/schemas/FOO/metadata-1.0.0
```

Therefore, in our schema file, we have the following keys, one declaring the name of the YAML `tag`, and one defining the `id` of the schema:

```
tag: "tag:stsci.edu:FOO/metadata-1.0.0"
id: "http://stsci.edu/schemas/FOO/metadata-1.0.0"
```

6.3.3 Descriptive information

Each schema has some descriptive fields: `title`, `description` and `examples`. These fields may contain core markdown syntax.

- `title`: A one-line summary of what the schema is for.
- `description`: A lengthier prose description of the schema
- `examples`: A list of example content that conforms to the schema, illustrating how to use it.

Continuing our example:

```
title: |
  Metadata for the FOO instrument.
description: |
  This stores some information about an exposure from the FOO instrument.
examples:
  -
    - A minimal description of an exposure.
    - |
        !FOO/metadata-1.0.0
          exposure_time: 0.001
```

6.3.4 The schema proper

The rest of the schema describes the acceptable data types and their structure. The format used for this description comes straight out of JSON Schema, and rather than documenting all of the things it can do here, please refer to Understanding JSON Schema (http://spacetelescope.github.io/understanding-json-schema/), and the further resources available at json-schema.org (http://json-schema.org).

In our example, we'll define two metadata elements: the name of the investigator, and the exposure time, each of which also have a description:

```
type: object
properties:
  investigator:
    type: string
    description: |
      The name of the principal investigator who requested the
      exposure.

  exposure_time:
    type: number
    description: |
      The time of the exposure, in nanoseconds.
```

We'll also define an optional element for the exposure time unit. This is a somewhat contrived example to demonstrate how to include elements in your schema that are based on the custom types defined in the ASDF standard:

```
exposure_time_units:
  $ref: "http://stsci.edu/schemas/asdf/unit/unit-1.0.0"
  description: |
    The unit of the exposure time.
  default:
    s
```

Lastly, we'll declare `exposure_time` as being required, and allow extra elements to be added:

```
required: [exposure_time]
additionalProperties: true
```

6.3.5 The complete example

Here is our complete schema example:

```
%YAML 1.1
---
$schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
tag: "tag:stsci.edu:FOO/metadata-1.0.0"
id: "http://stsci.edu/schemas/FOO/metadata-1.0.0"

title: |
  Metadata for the FOO instrument.
description: |
  This stores some information about an exposure from the FOO instrument.
examples:
  -
    - A minimal description of an exposure.
    - |
```

```
        !FOO/metadata-1.0.0
          exposure_time: 0.001

type: object
properties:
  investigator:
    type: string
    description: |
      The name of the principal investigator who requested the
      exposure.

  exposure_time:
    type: number
    description: |
      The time of the exposure, in nanoseconds.

  exposure_time_units:
    $ref: "http://stsci.edu/schemas/asdf/unit/unit-1.0.0"
    description: |
      The unit of the exposure time.
    default:
      s

required: [exposure_time]
additionalProperties: true
```

6.4 Extending an existing schema

TODO

CHAPTER 7

KNOWN LIMITS

The following is a catalogue of known limits in ASDF 1.0.0.

7.1 Tree

While there is no hard limit on the size of the Tree, in most practical implementations it will need to be read entirely into main memory in order to interpret it, particularly to support forward references. This imposes a practical limit on its size relative to the system memory on the machine. It is not recommended to store large data sets in the tree directly, instead it should reference blocks.

7.2 Literal integer values in the Tree

Different programming languages deal with numbers differently. For example, Python has arbitrary-length integers, while Javascript stores all numbers as 64-bit double-precision floats. It may be possible to write long integers from Python into the Tree, and upon reading in Javascript have undefined loss of information when reading those values back in.

Therefore, for practical reasons, integer literals in the Tree must be at most 52-bits.

7.3 Blocks

The maximum size of a block header is 65536 bytes.

Since the size of the block is stored in a 64-bit unsigned integer, the largest possible block size is around 18 exabytes. It is likely that other limitations on file size, such as an operating system's filesystem limitations, will be met long before that.

CHAPTER 8

CHANGES

8.1 Version 1.0.0

First pre-release.

CHAPTER 9

APPENDIX A: EMBEDDING ASDF IN FITS

While ASDF is designed to replace all of the existing use cases of FITS, there will still be cases where files need to be produced in FITS. Even then, it would be nice to take advantage of the highly-structured nature of ASDF to store content that can not easily be represented in FITS in a FITS file. This appendix describes a convention for embedding ASDF content in a FITS file.

The content of the ASDF file is placed in the data portion of an extra image extension named `ASDF` (`EXTNAME = 'ASDF'`). (By convention, the datatype is unsigned 8-bit integers (`BITPIX = 8`) and is one-dimensional (`NAXIS = 1`), but this is not strictly necessary.)

Rather than including a copy of the large data arrays in the ASDF extension, the ASDF content may refer to binary data stored in regular FITS extensions elsewhere in the same file. The convention for doing this is to set the `source` property of a *ndarray* (page 20) object to a special string identifier for a FITS reference. These values come in two forms:

- `fits:EXTNAME,EXTVER`: Where `EXTNAME` and `EXTVER` uniquely identify a FITS extension.
- `fits:INDEX`: Where `INDEX` is the zero-based index of a FITS extension.

The `fits:EXTNAME,EXTVER` form is preferred, since it allows for rearranging the FITS extensions in the file without the need to update the content of the `ASDF` extension, and thus such rearrangements could be performed by a non-ASDF-aware FITS library.

Such "FITS references" simply point to the binary content of the data portion of a FITS header/data unit. There is no enforcement that the `datatype` of the ASDF *ndarray* (page 20) matches the `BITPIX` of the FITS extension, or expectation that an explicit conversion would be performed if they don't match. It is up to the writer of the file to keep the ASDF and FITS datatype descriptions in sync.

The following is a schematic of an example FITS file with an ASDF extension. The ASDF content references the binary data in two FITS extensions elsewhere in the file.

```
HDU 0:
┌──────────────────────────────────────┐
|SIMPLE  = T                           |
|BITPIX  = -64                         |
|NAXIS   = 2                           |
|NAXIS1  = 512                         |
|NAXIS2  = 512                         |
|EXTEND  = T                           |
|EXTNAME = 'SCI     '                  |
|END                                   |
├──────────────────────────────────────┤
|...data...                            |<--┐
└──────────────────────────────────────┘   |
HDU 1:                                     |
┌──────────────────────────────────────┐   |
|XTENSION= 'IMAGE   '                  |   |
```

```
|BITPIX  = -64                        |    |
|NAXIS   = 2                          |    |
|NAXIS1  = 512                        |    |
|NAXIS2  = 512                        |    |
|EXTNAME = 'DQ      '                 |    |
|END                                  |    |
├-------------------------------------┤    |
|...data...                           |<--┼---┐
└-------------------------------------┘    |   |
HDU 2:                                     |   |
┌-------------------------------------┐    |   |
|XTENSION= 'IMAGE   '                 |    |   |
|BITPIX  = 8                          |    |   |
|NAXIS   = 1                          |    |   |
|NAXIS1  = 361                        |    |   |
|EXTNAME = 'ASDF    '                 |    |   |
|END                                  |    |   |
├-------------------------------------┤    |   |
|#ASDF 1.0.0                          |    |   |
|%YAML 1.1                            |    |   |
|%TAG ! tag:stsci.edu:asdf/           |    |   |
|--- !core/asdf-1.0.0                 |    |   |
|model:                               |    |   |
|  sci:                               |    |   |
|    data: !core/ndarray-1.0.0        |    |   |
|      source: fits:SCI,1   ----------┼---┘   |
|      datatype: float64              |       |
|      byteorder: little              |       |
|      shape: [512]                   |       |
|    wcs: ...WCS info...              |       |
|  dq:                                |       |
|    data: !core/ndarray-1.0.0        |       |
|      source: fits:DQ,1    ----------┼-------┘
|      datatype: float64              |
|      byteorder: little              |
|      shape: [512]                   |
|    wcs: ...WCS info...              |
|...                                  |
└-------------------------------------┘
```

A paper, ASDF: A new data format for astronomy (http://dx.doi.org/10.1016/j.ascom.2015.06.004) about ASDF has been published in Astronomy and Computing:

> Greenfield, P., Droettboom, M., & Bray, E. (2015). ASDF: A new data format for Astronomy. *Astronomy and Computing*. (In press). doi:10.1016/j.ascom.2015.06.004

[Thomas2015] Thomas, B., Jenness. T. et al. 2015, "The Future of Astronomical Data Formats I. Learning from FITS". Astronomy & Computing, in press, arXiv e-print: 1502.00996. https://github.com/timj/aandc-fits.