

ASDF Standard
Release 1.0.0

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

CONTENTS

1	1.1 Implementations	3 4
2	2.1 Header	7 8
3	The tree in-depth 3.1 Tags 1 3.2 References 1 3.3 Numeric literals 1 3.4 Comments 1	.3
4	Versioning Conventions14.1 Relationship of version numbers14.2 Handling version mismatches1	.5
5	ASDF schema definitions 1 5.1 Core 1 5.2 FITS 2 5.3 Unit 4 5.4 Time 4 5.5 Transform 5 5.6 WCS 12	11 16 18
6	Extending ASDF 14 6.1 YAML Schema 14 6.2 ASDF Schema 14 6.3 Designing a new tag and schema 14 6.4 Extending an existing schema 15	41 16
7	Known limits 15 7.1 Tree 7.2 Literal integer values in the Tree	53

	.3 Blocks	53
	Changes 1 .1 Version 1.0.0 .1	55
9	ppendix A: Embedding ASDF in FITS	57
Βi	ography 1	59

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.

CONTENTS 1

2 CONTENTS

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 (YAMLTM) 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 ") or UNIX (" \r ") 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 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. Blocks 7

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

2.6. Exploded form 9

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 ' \sim ' (0x7E) and '/' (0x2F) have special meanings, ' \sim ' needs to be encoded as ' \sim 0' and '/' needs to be encoded as ' \sim 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:

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:

3.3. Numeric literals

```
# 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 1.1
2
    $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: |
8
     This schema contains the top-level attributes for every ASDF file.
10
    tag: "tag:stsci.edu:asdf/core/asdf-1.0.0"
11
    type: object
12
    properties:
13
     asdf_library:
        description: |
15
          Describes the ASDF library that produced the file.
16
        $ref: "software-1.0.0"
17
18
      history:
19
        description: |
20
          A log of transformations that have happened to the file. May
21
          include such things as data collection, data calibration
22
          pipelines, data analysis etc.
23
        type: array
24
        items:
25
          $ref: "history_entry-1.0.0"
26
27
      data:
28
        description: |
29
          The data array corresponds to the main science data array in the
30
          file. Oftentimes, the data model will be much more complex than
31
          a single array, but this array will be used by applications that
32
          just want to convert to a display an image or preview of the
33
          file. It is recommended, but not required, that it is a
          2-dimensional image array.
35
        $ref: "ndarray-1.0.0"
36
37
38
      fits:
        description: |
39
          A way to specify exactly how this ASDF file should be converted
40
41
        $ref: "../fits/fits-1.0.0"
42
43
      wcs:
44
        description: |
45
          The location of the main WCS for the main data.
        $ref: "../wcs/wcs-1.0.0"
```

```
48 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
2
    $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: |
6
     Represents a complex number matching the following EBNF grammar
8
9
       plus-or-minus = "+" | "-"
10
                 = "J" | "j" | "I" | "i"
       suffix
11
       complex
                      = [ieee754] [plus-or-minus ieee754 suffix]
12
13
14
      Where `ieee754` is a floating point number in IEEE 754 decimal
15
      format.
16
17
      Though 'J', 'j', 'I' and 'i' must be supported on reading, it is
18
      recommended to use `i` on writing.
19
```

```
examples:
21
22
         - 1 real, -1 imaginary
23
          "!core/complex-1.0.0 1-1;"
2.4
25
        - 0 real, 1 imaginary
26
         - "!core/complex-1.0.0 1J"
27
28
         - -1 real, 0 imaginary
29
         - "!core/complex-1.0.0 -1"
30
31
    tag: "tag:stsci.edu:asdf/core/complex-1.0.0"
32
    type: string
    pattern: "([-+]?[0-9]*\\ \land ?[0-9]+([eE][-+]?[0-9]+)?)?([-+][0-9]*\\ \land .?[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 \geq 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
```

5.1. Core 21

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

Type: string from ["big", "little"].

byteorder

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

```
shape
```

```
Type: array of (integer \geq 0).
```

Items:

```
Type: integer \geq 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 \geq 0.

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

Default: 0

strides

Type: array of (integer \geq 1 or integer \leq -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 \geq 1 or integer \leq -1.

Any of:

Type: integer \geq 1.

Type: integer \leq -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:

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 1.1
1
2
    $schema: "http://stsci.edu/schemas/asdf/asdf-schema-1.0.0"
3
    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.
8
    description: |
10
      There are two ways to store the data in an ndarray.
11
12
      - Inline in the tree: This is recommended only for small arrays. In
13
        this case, the entire ``ndarray`` tag may be a nested list, in
14
        which case the type of the array is inferred from the content.
15
        (See the rules for type inference in the ``inline-data``
16
        definition below.) The inline data may also be given in the
17
        ``data`` property, in which case it is possible to explicitly
18
        specify the ``datatype`` and other properties.
19
20
      - External to the tree: The data comes from a [block](ref:block)
21
        within the same ASDF file or an external ASDF file referenced by a
22
23
24
    examples:
25
26
        - An inline array, with implicit data type
27
28
            !core/ndarray-1.0.0
29
              [[1, 0, 0],
30
               [0, 1, 0],
31
               [0, 0, 1]]
32
33
34
        - An inline array, with an explicit data type
35
36
            !core/ndarray-1.0.0
37
              datatype: float64
38
              data:
39
                [[1, 0, 0],
40
                  [0, 1, 0],
41
                  [0, 0, 1]]
42
43
44
        - An inline structured array, where the types of each column are
45
          automatically detected
        - 1
47
            !core/ndarray-1.0.0
48
              [[M110, 110, 205, And],
49
               [ M31, 31, 224, And],
50
               [ M32, 32, 221, And],
51
               [M103, 103, 581, Cas]]
52
53
54
        - An inline structured array, where the types of each column are
55
          explicitly specified
```

```
57
             !core/ndarray-1.0.0
58
               datatype: [['ascii', 4], uint16, uint16, ['ascii', 4]]
59
60
                 [[M110, 110, 205, And],
61
                  [ M31, 31, 224, And],
62
                   [ M32, 32, 221, And],
63
                   [M103, 103, 581, Cas]]
64
65
66
         - A double-precision array, in contiguous memory in a block within
67
           the same file
68
69
             !core/ndarray-1.0.0
70
               source: 0
71
               shape: [1024, 1024]
72
               datatype: float64
73
               byteorder: little
74
75
76
         - A view of a tile in that image
77
78
             !core/ndarray-1.0.0
79
               source: 0
80
               shape: [256, 256]
81
               datatype: float64
82
               byteorder: little
83
               strides: [8192, 8]
84
               offset: 2099200
85
86
87
         - A structured datatype, with nested columns for a coordinate in
88
           (*ra*, *dec*), and a 3x3 convolution kernel
89
90
             !core/ndarray-1.0.0
91
               source: 0
92
               shape: [64]
93
94
               datatype:
                 - name: coordinate
95
                   datatype:
96
                     - name: ra
97
                        datatype: float64
98
                      - name: dec
99
                        datatype: float64
100
                  - name: kernel
101
                    datatype: float32
102
                    shape: [3, 3]
103
               byteorder: little
104
105
106
         - An array in Fortran order
107
             !core/ndarray-1.0.0
109
               source: 0
110
               shape: [1024, 1024]
111
               datatype: float64
112
               byteorder: little
113
               strides: [8192, 8]
114
```

```
116
         - An array where values of -999 are treated as missing
117
118
             !core/ndarray-1.0.0
119
               source: 0
120
               shape: [256, 256]
121
               datatype: float64
122
               byteorder: little
123
               mask: -999
124
125
126
         - An array where another array is used as a mask
127
128
             !core/ndarray-1.0.0
129
               source: 0
130
               shape: [256, 256]
131
               datatype: float64
132
               byteorder: little
133
               mask: !core/ndarray-1.0.0
134
                 source: 1
135
                 shape: [256, 256]
136
                 datatype: bool8
137
                 byteorder: little
138
139
140
141
         - An array where the data is stored in the first block in
142
           another ASDF file.
143
             !core/ndarray-1.0.0
144
               source: external.asdf
145
               shape: [256, 256]
               datatype: float64
               byteorder: little
148
149
    definitions:
150
      scalar-datatype:
151
         description: |
152
           Describes the type of a single element.
153
155
           There is a set of numeric types, each with a single identifier:
156
           - `int8`, `int16`, `int32`, `int64`: Signed integer types, with
157
             the given bit size.
158
159
           - `uint8`, `uint16`, `uint32`, `uint64`: Unsigned integer types,
             with the given bit size.
161
162
           - `float32`: Single-precision floating-point type or "binary32",
163
             as defined in IEEE 754.
164
165
           - `float64`: Double-precision floating-point type or "binary64",
             as defined in IEEE 754.
168
           - `complex64`: Complex number where the real and imaginary parts
169
             are each single-precision floating-point ("binary32") numbers,
170
             as defined in IEEE 754.
171
172
```

```
- `complex128`: Complex number where the real and imaginary
173
             parts are each double-precision floating-point ("binary64")
174
             numbers, as defined in IEEE 754.
175
176
           There are two distinct fixed-length string types, which must
177
           be indicated with a 2-element array where the first element is an
           identifier for the string type, and the second is a length:
179
180
           - `ascii`: A string containing ASCII text (all codepoints <
181
             128), where each character is 1 byte.
182
183
           - `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.
186
187
         anyOf:
188
           - type: string
189
             enum: [int8, uint8, int16, uint16, int32, uint32, int64, uint64,
190
                     float32, float64, complex64, complex128, bool8]
191
           - type: array
192
             items:
193
               - type: string
194
                 enum: [ascii, ucs4]
195
               - type: integer
196
                 minimum: 0
             minLength: 2
             maxLength: 2
199
200
       datatype:
201
         description: |
202
           The data format of the array elements. May be a single scalar
203
           datatype, or may be a nested list of datatypes. When a list, each field
204
           may have a name.
205
         anyOf:
206
           - $ref: "#/definitions/scalar-datatype"
207
           - type: array
208
             items:
209
               anyOf:
210
                 - $ref: "#/definitions/scalar-datatype"
                 - type: object
212
                   properties:
213
                     name:
214
                        type: string
215
                        pattern: "[A-Za-z_][A-Za-z0-9_]*"
216
                        description: The name of the field
217
                     datatype:
218
                        $ref: "#/definitions/datatype"
219
                     byteorder:
220
                        type: string
221
                        enum: [big, little]
222
223
                        description: |
                          The byteorder for the field. If not provided, the
                          byteorder of the datatype as a whole will be used.
225
                      shape:
226
                        type: array
227
                        items:
228
                          type: integer
229
                          minimum: 0
```

```
required: [datatype]
231
232
       inline-data:
233
         description: |
234
           Inline data is stored in YAML format directly in the tree, rather than
235
           referencing a binary block. It is made out of nested lists.
236
237
           If the datatype of the array is not specified, it is inferred from
238
           the array contents. Type inference is supported only for
239
           homogeneous arrays, not tables.
240
241
           - If any of the elements in the array are YAML strings, the
242
             'datatype' of the entire array is 'ucs4', with the width of
             the largest string in the column, otherwise...
244
245
           - If any of the elements in the array are complex numbers, the
246
             'datatype' of the entire column is 'complex128', otherwise...
247
248
           - If any of the types in the column are numbers with a decimal
249
             point, the 'datatype' of the entire column is 'float64',
250
             otherwise..
251
252
           - If any of the types in the column are integers, the `datatype`
253
            of the entire column is `int64`, otherwise...
254
255
           - The `datatype` of the entire column is `bool8`.
           Masked values may be included in the array using `null`. If an
258
           explicit mask array is also provided, it takes precedence.
259
260
         type: array
261
         items:
262
           anyOf:
263
             - type: number
264
             - type: string
265
             - type: "null"
266
             - $ref: "complex-1.0.0"
267
             - $ref: "#/definitions/inline-data"
268
             - type: boolean
270
    anyOf:
271
      - $ref: "#/definitions/inline-data"
272
       - type: object
273
         properties:
274
           source:
275
             description: |
276
               The source of the data.
277
278
               - If an integer: If positive, the zero-based index of the
279
                 block within the same file. If negative, the index from
280
                 the last block within the same file. For example, a
281
                 source of `-1` corresponds to the last block in the same
                 file.
283
284
               - If a string, a URI to an external ASDF file containing the
285
                 block data. Relative URIs and ``file:`` and ``http:`
286
                 protocols must be supported. Other protocols may be supported
287
                 by specific library implementations.
```

```
The ability to reference block data in an external ASDF file
290
               is intentionally limited to the first block in the external
291
               ASDF file, and is intended only to support the needs of
292
               [exploded](ref:exploded). For the more general case of
293
               referencing data in an external ASDF file, use tree
               [references](ref:references).
295
296
             anyOf:
297
               - type: integer
298
               - type: string
299
                 format: uri
300
301
302
           data:
             description: |
303
               The data for the array inline.
304
305
               If 'datatype' and/or 'shape' are also provided, they must
306
               match the data here and can be used as a consistency check.
307
                `strides`, `offset` and `byteorder` are meaningless when
308
                `data` is provided.
309
             $ref: "#/definitions/inline-data"
310
311
           shape:
312
             description: |
313
               The shape of the array.
314
315
316
               The first entry may be the string `*`, indicating that the
               length of the first index of the array will be automatically
317
               determined from the size of the block. This is used for
318
               streaming support.
319
             type: array
             items:
               anyOf:
322
                  - type: integer
323
                    minimum: 0
324
                  - enum: ['*']
325
326
327
           datatype:
             description: |
               The data format of the array elements.
329
             $ref: "#/definitions/datatype"
330
331
           byteorder:
332
             description: >
333
               The byte order (big- or little-endian) of the array data.
             type: string
335
             enum: [big, little]
336
337
           offset:
338
             description: >
339
               The offset, in bytes, within the data for this start of this
340
               view.
341
             type: integer
342
             minimum: 0
343
             default: 0
344
345
           strides:
346
```

```
description: >
347
               The number of bytes to skip in each dimension. If not provided,
348
               the array is assumed by be contiguous and in C order.
349
               provided, must be the same length as the shape property.
350
             type: array
351
             items:
352
               anyOf:
353
                 - type: integer
354
                   minimum: 1
355
                 - type: integer
356
                   maximum: -1
357
           mask:
359
             description: >
360
               Describes how missing values in the array are stored. If a
361
               scalar number, that number is used to represent missing values.
362
               If an ndarray, the given array provides a mask, where non-zero
363
               values represent missing values in this array. The mask array
364
               must be broadcastable to the dimensions of this array.
365
             anyOf:
366
               - type: number
367
               - $ref: "complex-1.0.0"
368
               - allOf:
369
                 - $ref: "ndarray-1.0.0"
370
                 - datatype: bool8
         dependencies:
373
           source: [shape, datatype, byteorder]
374
375
         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.
```

5.1. Core 33

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
```

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
```

```
%YAML 1.1
2
    $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)
12
      object, containing the data and some additional information.
13
14
      The data itself may be stored inline as text, or in binary in either
15
      row- or column-major order by use of the `strides` property on the
      individual column arrays.
18
      Each column in the table must have the same first (slowest moving)
19
      dimension.
20
21
    examples:
22
23
        - A table stored in column-major order, with each column in a separate block
25
            !core/table-1.0.0
26
              columns:
27
              - !core/column-1.0.0
28
                data: !core/ndarray-1.0.0
29
                  source: 0
                  datatype: float64
31
                  byteorder: little
32
                  shape: [3]
33
                description: RA
34
                meta: {foo: bar}
35
36
                name: a
                unit: !unit/unit-1.0.0 deg
37
              -!core/column-1.0.0
38
                data: !core/ndarray-1.0.0
39
                  source: 1
40
                  datatype: float64
```

5.1. Core 35

```
byteorder: little
42
                   shape: [3]
43
                 description: DEC
44
                 name: b
45
               - !core/column-1.0.0
46
                 data: !core/ndarray-1.0.0
47
                   source: 2
48
                   datatype: [ascii, 1]
49
                   byteorder: big
50
                   shape: [3]
51
                 description: The target name
52
53
                 name: c
54
55
        - A table stored in row-major order, all stored in the same block
56
57
             !core/table-1.0.0
58
               columns:
59
               - !core/column-1.0.0
60
                 data: !core/ndarray-1.0.0
61
                   source: 0
62
                   datatype: float64
63
                   byteorder: little
64
                   shape: [3]
65
                   strides: [13]
66
                 description: RA
67
                 meta: {foo: bar}
68
                 name: a
69
                 unit: !unit/unit-1.0.0 deg
70
               -!core/column-1.0.0
71
                 data: !core/ndarray-1.0.0
72
                   source: 0
73
                   datatype: float64
                   byteorder: little
75
                   shape: [3]
76
                   offset: 4
77
                   strides: [13]
78
                 description: DEC
                 name: b
80
               - !core/column-1.0.0
81
                 data: !core/ndarray-1.0.0
82
                   source: 0
83
                   datatype: [ascii, 1]
84
                   byteorder: big
85
                   shape: [3]
86
                   offset: 12
87
                   strides: [13]
88
                 description: The target name
89
                 name: c
90
91
    type: object
92
    properties:
93
      columns:
        description: |
95
          A list of columns in the table.
96
        type: array
97
        items:
98
          $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
2
3 $schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
4 id: "http://stsci.edu/schemas/asdf/core/column-1.0.0"
5 tag: "tag:stsci.edu:asdf/core/column-1.0.0"
```

5.1. Core 37

```
title: >
      A column in a table.
8
    description: |
     Each column contains a name and an array of data, and an optional description
11
      and unit.
12
13
    type: object
14
    properties:
15
      name:
16
        description: |
17
          The name of the column. Each name in a
18
          [table](http://stsci.edu/schemas/asdf/core/table-1.0.0) must be
19
          unique.
20
        type: string
21
        pattern: "[A-Za-z_][A-Za-z0-9_]*"
22
23
24
      data:
        description: |
25
          The array data for the column.
26
        allOf:
2.7
          - $ref: ndarray-1.0.0
28
29
      description:
30
31
        description: |
          An optional description of the column.
32
        type: string
33
        default: ''
34
35
      unit:
36
37
        description:
          An optional unit for the column.
38
        allOf:
39
          - $ref: ../unit/unit-1.0.0
40
41
      meta:
42
        description:
43
          Additional free-form metadata about the column.
45
        type: object
46
        default: {}
47
    requiredProperties: [name, data]
48
    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.

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
2
   $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:
10
11
     name:
        description: |
12
          The name of the application or library.
13
        type: string
```

5.1. Core 39

```
15
      author:
16
        description: |
17
          The author (or institution) that produced the software package.
18
        type: string
19
20
      homepage:
21
        description: |
22
          A URI to the homepage of the software.
23
        type: string
24
        format: uri
25
26
27
      version:
28
        description: |
29
          The version of the software used. It is recommended, but not
          required, that this follows the (Semantic Versioning
30
          Specification)[http://semver.org/spec/v2.0.0.html].
31
        type: string
32
33
    required: [name, author, homepage, version]
34
    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).
```

```
%YAML 1.1
1
2
    $schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
3
    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
9
    properties:
10
      description:
11
        description: |
12
          A description of the transformation performed.
13
        type: string
14
15
      time:
16
        description: |
17
          A timestamp for the operation, in UTC.
18
        type: string
19
        format: date-time
20
21
      software:
22
        description: |
23
          One or more descriptions of the software that performed the
24
          operation.
25
        anyOf:
          - $ref: "software-1.0.0"
27
          - type: array
28
            items:
29
               $ref: "software-1.0.0"
30
31
    requiredProperties: [description]
32
    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.

5.2. FITS 41

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 \le len \le 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 \le len \le 3.
       Items:
           index[0]
           Type: string ( len \le 8 \text{ regex } [A-Z0-9]* ).
           The keyword.
           index[1]
           Type: string ( len \le 60 ) or number or boolean.
           The value.
          Any of:
              Type: string ( len \le 60 ).
              Type: number.
              Type: boolean.
           index[2]
           Type: string ( len \le 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]
```

5.2. FITS 43

```
%YAML 1.1
1
2
    $schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
3
    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
8
      automatically be converted to FITS files by specifying the exact
9
      content of the resulting FITS file.
10
11
      Not all kinds of data in FITS are directly representable in ASDF.
12
      For example, applying an offset and scale to the data using the
13
      `BZERO` and `BSCALE` keywords. In these cases, it will not be
14
      possible to store the data in the native format from FITS and also
15
      be accessible in its proper form in the ASDF file.
16
17
      Only image and binary table extensions are supported.
18
19
20
    examples:
21
        - A simple FITS file with a primary header and two extensions
22
23
            !fits/fits-1.0.0
24
                 - header:
25
                  - [SIMPLE, true, conforms to FITS standard]
                  - [BITPIX, 8, array data type]
27
                  - [NAXIS, 0, number of array dimensions]
28
                  - [EXTEND, true]
29
                  - []
30
                  - ['', Top Level MIRI Metadata]
31
                  - []
32
                  - [DATE, '2013-08-30T10:49:55.070373', The date this file was created (UTC)]
33
                  - [FILENAME, MiriDarkReferenceModel_test.fits, The name of the file]
34
                  - [TELESCOP, JWST, The telescope used to acquire the data]
35
                  - []
36
                  - ['', Information about the observation]
37
                  - []
38
                  - [DATE-OBS, '2013-08-30T10:49:55.000000', The date the observation was made (UTC)]
39
                 - data: !core/ndarray-1.0.0
40
                    datatype: float32
41
                    shape: [2, 3, 3, 4]
42
                    source: 0
43
                    byteorder: big
44
45
                  header:
                  - [XTENSION, IMAGE, Image extension]
                  - [BITPIX, -32, array data type]
                  - [NAXIS, 4, number of array dimensions]
48
                  - [NAXIS1, 4]
49
                  - [NAXIS2, 3]
50
                  - [NAXIS3, 3]
51
                  - [NAXIS4, 2]
52
                  - [PCOUNT, 0, number of parameters]
53
                  - [GCOUNT, 1, number of groups]
54
                  - [EXTNAME, SCI, extension name]
55
                  - [BUNIT, DN, Units of the data array]
56
```

```
- data: !core/ndarray-1.0.0
57
                      datatype: float32
58
                      shape: [2, 3, 3, 4]
59
                      source: 1
60
                      byteorder: big
61
                   header:
62
                    - [XTENSION, IMAGE, Image extension]
63
                    - [BITPIX, -32, array data type]
64
                    - [NAXIS, 4, number of array dimensions]
65
                    - [NAXIS1, 4]
66
                    - [NAXIS2, 3]
67
                    - [NAXIS3, 3]
68
                    - [NAXIS4, 2]
69
                    - [PCOUNT, 0, number of parameters]
70
                    - [GCOUNT, 1, number of groups]
71
                    - [EXTNAME, ERR, extension name]
72
                    - [BUNIT, DN, Units of the error array]
73
74
    tag: "tag:stsci.edu:asdf/fits/fits-1.0.0"
    type: array
76
    items:
77
      description: >
78
         Each item represents a single header/data unit (HDU).
79
       type: object
80
      properties:
81
         header:
82
           description: >
83
             A list of the keyword/value/comment triples from the header,
84
             in the order they appear in the FITS file.
85
           type: array
86
           items:
87
             type: array
88
             minItems: 0
89
             maxItems: 3
90
             items:
91
               - description: "The keyword."
92
                 type: string
93
                 maxLength: 8
94
                 pattern: "[A-Z0-9]*"
95
               - description: "The value."
96
                 anyOf:
97
                    - type: string
98
                      maxLength: 60
99
                    - type: number
100
                    - type: boolean
101
               - description: "The comment."
102
                 type: string
103
                 maxLength: 60
104
         data:
105
           description: "The data part of the HDU."
106
           anyOf:
107
             - $ref: "../core/ndarray-1.0.0"
             - $ref: "../core/table-1.0.0"
109
             - type: "null"
110
           default: null
111
       required: [header]
112
       additionalProperties: false
113
```

5.2. FITS 45

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 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
10
11
      in VOUnit syntax.
    examples:
12
13
        - Example unit
14
15
          !unit/unit-1.0.0 "2.1798721 10-18kg m2 s-2"
16
17
    anyOf:
18
      - tag: "tag:stsci.edu:asdf/unit/unit-1.0.0"
19
      - {}
20
21
    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 1.1
2
    $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: |
6
     Defines a new unit. It can be used to either:
      - Define a new base unit.
Q
10
      - Create a new unit name that is a equivalent to a given unit.
11
12
      The new unit must be defined before any unit tags that use it.
13
14
    tag: "tag:stsci.edu:asdf/unit/defunit-1.0.0"
15
    type: object
16
    properties:
     name:
18
        description: The name of the new unit.
19
        type: string
20
        pattern: "[A-Za-z_][A-Za-z0-9_]+"
21
```

5.3. Unit 47

```
unit:
23
        description: |
24
          The unit that the new name is equivalent to. It is optional,
25
          and if not provided, or ``null``, this ``defunit`` defines a new
26
          base unit.
27
28
        anyOf:
29
          - $ref: "unit-1.0.0"
30
          - type: "null"
31
32
    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})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9})(3[0-5]_{0-9}
          ).
          string_formats
          Type: definitions/iso time (page 48) or definitions/byear (page 48) or definitions/jyear (page 48) or definitions/
          tions/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)).
                    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.
          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", "jear", "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:

```
Х
```

Type: number. Required.

У

Type: number. Required.

Z

Type: number. Required.

unit

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

All of:

```
Type: unit-1.0.0 (page 46).
       Type: any.
       Default: "m"
Type: object.
Properties:
   long
   Type: number -180 \le x \le 180. Required.
   lat
   Type: number -90 \leq x \leq 90. Required.
   Type: number.
   Default: 0
   unit
   Type: unit-1.0.0 (page 46) and any.
   All of:
       Type: unit-1.0.0 (page 46).
       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
2
    $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: |
6
     A "time" is a single instant in time. It may explicitly specify the
      way time is represented (the "format") and the "scale" which
8
      specifies the offset and scaling relation of the unit of time.
Q
10
      Specific emphasis is placed on supporting time scales (e.g. UTC,
11
      TAI, UT1, TDB) and time representations (e.g. JD, MJD, ISO 8601)
      that are used in astronomy and required to calculate, e.g., sidereal
13
      times and barycentric corrections.
14
15
      Times may be represented as one of the following:
16
17
      - an object, with explicit 'value', and optional 'format', 'scale'
18
        and `location`.
19
20
      - a string, in which case the format is guessed from across
21
        the unambiguous options ('iso', 'byear', 'jyear', 'yday'), and the
22
        scale is hardcoded to `UTC`.
23
24
      In either case, a single time tag may be used to represent an
      n-dimensional array of times, using either an 'ndarray' tag or
26
      inline as (possibly nested) YAML lists. If YAML lists, the same
27
      format must be used for all time values.
28
29
      The precision of the numeric formats should only be assumed to be as
30
      good as an IEEE-754 double precision (float64) value. If
31
      higher-precision is required, the `iso` or `yday` format should be
```

```
used.
33
34
    examples:
35
36
        - Example ISO time
37
38
            !time/time-1.0.0 "2000-12-31T13:05:27.737"
39
40
41
        - Example year, day-of-year and time format time
42
43
            !time/time-1.0.0 "2001:003:04:05:06.789"
44
45
46
        - Example Besselian Epoch time
47
48
            !time/time-1.0.0 B2000.0
49
50
51
        - Example Besselian Epoch time, equivalent to above
52
53
            !time/time-1.0.0
54
              value: 2000.0
55
              format: byear
56
57
58
        - Example list of times
59
60
            !time/time-1.0.0
61
              ["2000-12-31T13:05:27.737", "2000-12-31T13:06:38.444"]
62
63
64
        - Example of an array of times
65
66
            !time/time-1.0.0
67
              value: !core/ndarray-1.0.0
68
                data: [2000, 2001]
69
                datatype: float64
70
              format: jyear
71
72
73
        - Example with a location
74
75
            !time/time-1.0.0
76
              value: 2000.0
77
              format: jyear
78
              scale: tdb
79
              location:
80
                x: 6378100
81
                y: 0
82
                z: 0
83
    definitions:
85
      iso_time:
86
        type: string
87
        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](.|
88
89
      byear:
```

```
type: string
91
        pattern: "B[0-9]+(.[0-9]+)?"
92
93
      jyear:
94
        type: string
95
        pattern: "J[0-9]+(.[0-9]+)?"
96
      yday:
98
        type: string
99
        100
101
102
      string_formats:
103
        anyOf:
          - "#/definitions/iso_time"
104
          - $ref: "#/definitions/byear"
105
          - $ref: "#/definitions/jyear"
106
          - $ref: "#/definitions/yday"
107
108
      array_of_strings:
109
        type: array
110
        items:
111
          anyOf:
112
            - $ref: "#/definitions/array_of_strings"
113
            - $ref: "#/definitions/string_formats"
114
115
    allOf:
      - tag: "tag:stsci.edu:asdf/time/time-1.0.0"
117
      - anyOf:
118
        - $ref: "#/definitions/string_formats"
119
120
        - $ref: "#/definitions/array_of_strings"
121
122
        - type: object
123
          properties:
124
            $ref: "../core/ndarray-1.0.0#anyOf/1/properties"
125
126
        - type: object
127
          properties:
128
            value:
129
              description: |
130
                The value(s) of the time.
131
132
              anyOf:
133
                - $ref: "#/definitions/string_formats"
134
                - "#/definitions/array_of_strings"
135
                - $ref: "../core/ndarray-1.0.0"
136
                - type: number
137
138
            format:
139
              description: |
140
                The format of the time.
141
                If not provided, the the format should be guessed from the
143
                string from among the following unambiguous options:
144
                `iso`, `byear`, `jyear` and `yday`.
145
146
                The supported formats are:
147
148
```

```
`iso`: ISO 8601 compliant date-time format
149
                    `YYYY-MM-DDTHH:MM:SS.sss...`. For example,
150
                   `2000-01-01 00:00:00.000` is midnight on January 1,
151
                   2000. The 'T' separating the date from the time
152
                   section is optional.
153
                  - `yday`: Year, day-of-year and time as
155
                   'YYYY:DOY:HH:MM:SS.sss...'. The day-of-year (DOY) goes
156
                   from 001 to 365 (366 in leap years). For example,
157
                   `2000:001:00:00:00.000` is midnight on January 1,
158
                   2000.
159
                 - `byear`: Besselian Epoch year, eg. `B1950.0`. The `B`
161
                   is optional if the 'byear' format is explicitly
162
                   specified.
163
164
                 - 'jyear': Julian Epoch year, eg. 'J2000.0'. The 'J' is
165
                   optional if the 'jyear' format is explicitly
166
                   specified.
168
                 - `decimalyear`: Time as a decimal year, with integer
169
                   values corresponding to midnight of the first day of
170
                   each year. For example 2000.5 corresponds to the ISO
171
                   time `2000-07-02 00:00:00`.
172
                 - `jd`: Julian Date time format. This represents the
174
                   number of days since the beginning of the Julian
175
                   Period. For example, 2451544.5 in 'jd' is midnight on
176
                   January 1, 2000.
177
178
                 - `mjd`: Modified Julian Date time format. This
                   represents the number of days since midnight on
181
                   November 17, 1858. For example, 51544.0 in MJD is
                   midnight on January 1, 2000.
182
183
                 - 'gps': GPS time: seconds from 1980-01-06 00:00:00 UTC
184
                   For example, 630720013.0 is midnight on January 1,
185
                   2000.
                 - `unix`: Unix time: seconds from 1970-01-01 00:00:00
188
                   UTC. For example, 946684800.0 in Unix time is midnight
189
                   on January 1, 2000. [TODO: Astropy's definition of
190
                   UNIX time doesn't match POSIX's here. What should we
191
                   do for the purposes of ASDF?]
192
193
               enum:
194
                 - iso
195
                 - yday
196
                 - byear
197
                 - jyear
198
                 - decimalyear
                 - jd
                 - mjd
201
                 - gps
202
                 - unix
203
                 - cxcsec
204
205
             scale:
```

```
description: |
207
                 The time scale (or time standard) is a specification for
208
                 measuring time: either the rate at which time passes; or
209
                 points in time; or both. See also [3] and [4].
210
211
                 These scales are defined in detail in [SOFA Time Scale and
212
                 Calendar Tools](http://www.iausofa.org/sofa_ts_c.pdf).
213
214
                 The supported time scales are:
215
216
                 - `utc`: Coordinated Universal Time (UTC). This is the
217
                    default time scale, except for 'gps', 'unix'.
218
                 - `tai`: International Atomic Time (TAI).
220
221
                  - `tcb`: Barycentric Coordinate Time (TCB).
222
223
                 - `tcg`: Geocentric Coordinate Time (TCG).
224
225
                 - `tdb`: Barycentric Dynamical Time (TDB).
226
227
                 - `tt`: Terrestrial Time (TT).
228
229
                 - `ut1`: Universal Time (UT1).
230
231
               enum:
                 - utc
233
                  - tai
234
                  - tcb
235
                 - tcg
236
                 - tdb
237
                  - tt
238
                  - ut1
239
240
             location:
241
               description: |
242
                 Specifies the observer location for scales that are
243
                 sensitive to observer location, currently only `tdb`. May
244
                 be specified either with geocentric coordinates (X, Y, Z)
                 with an optional unit or geodetic coordinates:
246
                    - `long`: longitude in degrees
247
                    - `lat`: in degrees
248
                    - `h`: optional height
249
250
               anyOf:
251
                  - type: object
252
                    properties:
253
                      х:
254
                        type: number
255
                      у:
256
257
                        type: number
                      z:
                        type: number
259
                      unit:
260
                        allOf:
261
                          - $ref: "../unit/unit-1.0.0"
262
                          - default: m
263
                    required: [x, y, z]
264
```

```
- type: object
265
                     properties:
266
                       long:
267
                         type: number
268
                         minimum: -180
269
                         maximum: 180
271
                         type: number
272
                         minimum: -90
273
                         maximum: 90
274
275
                         type: number
                         default: 0
                       unit:
278
                         allOf:
279
                            - $ref: "../unit/unit-1.0.0"
280
                            - default: m
281
                     required: [long, lat]
282
283
           required: [value]
284
```

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.

Chapter 5. ASDF schema definitions

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 1.1
2
    $schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
3
    id: "http://stsci.edu/schemas/asdf/transform/transform-1.0.0"
     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.
10
11
    type: object
12
    properties:
13
      name:
14
        description: |
15
          A user-friendly name for the transform, to give it extra
16
          meaning.
17
        type: string
18
19
      domain:
20
        description: |
21
          The domain (range of valid inputs) to the transform.
22
          Each entry in the list corresponds to an input dimension.
23
        type: array
24
        items:
25
          $ref: "domain-1.0.0"
26
27
      inverse:
28
        description: |
29
          Explicitly sets the inverse transform of this transform.
30
31
          If the transform has a direct analytic inverse, this
32
          property is usually not necessary, as the ASDF-reading tool
33
          can provide it automatically.
34
35
        $ref: "transform-1.0.0"
36
    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**.

5.5. Transform 59

```
All of:

0
Type: transform-1.0.0 (page 58).

1
Type: object.

Properties:

n_inputs

Type: integer. Required.

n_outputs

Type: integer. Required.
```

```
%YAML 1.1
2
   $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
10
     written. **IT WILL NOT BE IN THE FINAL VERSION OF THE SPEC**.
11
12
   allOf:
13
      - $ref: "transform-1.0.0"
14
      - type: object
15
       properties:
16
         n_inputs:
17
           type: integer
19
          n_outputs:
            type: integer
20
        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
1
2
   $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.
10
   allOf:
11
     - $ref: "transform-1.0.0"
12
      - type: object
13
        properties:
14
          n_dims:
15
            type: integer
16
            default: 1
17
            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:

```
0Type: transform-1.0.0 (page 58).1Type: object.Properties:
    value
    Type: number. Required.
```

5.5. Transform 61

```
%YAML 1.1
2
   $schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
3
   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.
8
   description: |
Q
     Invertibility: All ASDF tools are required to be able to compute the
10
     analytic inverse of this transform, which always outputs zero values.
   allOf:
13
     - $ref: "transform-1.0.0"
      - type: object
14
        properties:
15
          value:
16
17
            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
2
   $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: >
6
     Defines the domain of an input axis.
8
   description: >
     Describes the range of acceptable input values to a particular
10
     axis of a transform.
11
   examples:
13
14
        - The domain `[0, 1)`.
15
```

```
!transform/domain-1.0.0
17
             lower: 0
18
            upper: 1
19
             includes_lower: true
20
21
22
    properties:
      lower:
23
        description: >
24
          The lower value of the domain. If not provided, the
25
          domain has no lower limit.
26
        type: number
27
        default: -.inf
28
29
      upper:
30
        description: >
31
          The upper value of the domain. If not provided, the
32
          domain has no upper limit.
33
        type: number
34
        default: .inf
35
36
      includes_lower:
37
        description: If 'true', the domain includes 'lower'.
38
        type: boolean
39
        default: false
40
42
      includes_upper:
        description: If `true`, the domain includes `upper`.
43
        type: boolean
44
        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:

```
0Type: transform-1.0.0 (page 58).1Type: any.
```

Examples:

A series of transforms:

5.5. Transform 63

```
!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
```

```
%YAML 1.1
2
    $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
10
      subtransform.
11
12
      The number of output dimensions of each subtransform must be equal
13
      to the number of input dimensions of the next subtransform in list.
14
      To reorder or add/drop axes, insert `remap_axes` transforms in the
15
      subtransform list.
16
      Invertibility: All ASDF tools are required to be able to compute the
18
      analytic inverse of this transform, by reversing the list of
19
      transforms and applying the inverse of each.
20
21
    examples:
22
23
        - A series of transforms
24
25
          !transform/compose-1.0.0
26
            forward:
27
              - !transform/generic-1.0.0
28
                n_inputs: 1
29
                n_outputs: 2
              - !transform/generic-1.0.0
31
                n_inputs: 2
32
                n_outputs: 1
33
34
    allOf:
35
      - $ref: "transform-1.0.0"
36
      - properties:
37
          forward:
38
            type: array
39
40
              $ref: "transform-1.0.0"
41
        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:

```
    transform A: 2 in -> 2 out
    transform B: 1 in -> 2 out
```

3. transform C: 2 in -> 1 out

The transform is performed as follows:

```
i0
   i1
        i2
В
             С
   \perp
   ο2
          о3
               о4
```

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:

```
0Type: transform-1.0.0 (page 58).1Type: 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: 1
```

5.5. Transform 65

```
%YAML 1.1
2
    $schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
3
    id: "http://stsci.edu/schemas/asdf/transform/concatenate-1.0.0"
    tag: "tag:stsci.edu:asdf/transform/concatenate-1.0.0"
    title: >
6
7
     Send axes to different subtransforms.
8
    description: |
9
     Transforms a set of separable inputs by splitting the axes apart,
10
      sending them through the given subtransforms in parallel, and
11
      finally concatenating the subtransform output axes back together.
12
      The input axes are assigned to each subtransform in order. If the
14
     number of input axes is unequal to the sum of the number of input
15
      axes of all of the subtransforms, that is considered an error case.
16
17
      The output axes from each subtransform are appended together to make
18
      up the resulting output axes.
19
20
      For example, given 5 input axes, and 3 subtransforms with the
21
      following orders:
22
23
      1. transform A: 2 in -> 2 out
24
      1. transform B: 1 in -> 2 out
25
      1. transform C: 2 in -> 1 out
26
27
      The transform is performed as follows:
28
29
30
            i0
                i1
                          i2
                                   i3 i4
31
            32
33
           | A | B | C
34
35
           36
            00 01 02 03
37
38
39
      If reordering of the input or output axes is required, use in series
40
     with the 'remap_axes' transform.
41
42
      Invertibility: All ASDF tools are required to be able to compute the
43
     analytic inverse of this transform.
44
    examples:
45
46
        - The example in the description
47
48
          !transform/concatenate-1.0.0
49
           forward:
50
             - !transform/generic-1.0.0
51
               n_inputs: 2
52
               n_outputs: 2
53
              - !transform/generic-1.0.0
54
               n_inputs: 1
55
               n_outputs: 2
```

```
- !transform/generic-1.0.0
57
                 n_inputs: 2
58
                 n_outputs: 1
59
60
    allOf:
61
      - $ref: "transform-1.0.0"
62
      - properties:
63
           forward:
64
             type: array
65
             items:
66
               $ref: "transform-1.0.0"
67
         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.

5.5. Transform 67

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
2
   $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: >
6
     Reorder, add and drop axes.
8
   description: |
9
     This transform allows the order of the input axes to be shuffled and
10
     returned as the output axes.
11
     It is a list made up of integers or "constant markers". Each item
13
     in the list corresponds to an output axis. For each item:
14
15
     - If an integer, it is the index of the input axis to send to the
```

```
output axis.
17
18
      - If a constant, it must be a single item which is a constant value
19
        to send to the output axis.
20
21
      If only a list is provided, the number of input axes is
22
      automatically determined from the maximum index in the list. If an
23
      object with 'mapping' and 'n_inputs' properties is provided, the
24
      number of input axes is explicitly set by the `n_inputs` value.
25
26
      Invertibility: TBD
27
    examples:
28
29
        - For 2 input axes, swap the axes
30
31
            !transform/remap_axes-1.0.0
32
              mapping: [1, 0]
33
34
        - For 2 input axes, return the second axis and drop the first
35
36
            !transform/remap_axes-1.0.0
37
              mapping: [1]
38
39
40
        - For 2 input axes, return the first axis twice, followed by the second
41
42
            !transform/remap_axes-1.0.0
43
              mapping: [0, 0, 1]
44
45
46
        - For 2 input axes, add a third axis which is a constant
47
48
            !transform/remap_axes-1.0.0
49
              mapping: [0, 1, !core/constant-1.0.0 42]
50
51
52
53
            The above example is equivalent to the following, and ASDF
54
            implementations are free to normalize it thusly:
55
56
            !transform/concatenate-1.0.0
57
              forward:
58
                - !transform/remap_axes-1.0.0
59
                  mapping: [0]
60
                - !transform/remap_axes-1.0.0
61
                  mapping: [1]
62
                - !transform/constant-1.0.0
63
                   value: 42
64
65
66
        - Here we have 3 input axes, but we are explicitly dropping the last one
67
68
            !transform/remap_axes-1.0.0
69
              mapping: [0, 1]
70
              n_inputs: 3
71
72
    definitions:
73
      mapping:
```

```
type: array
75
        items:
76
          anyOf:
77
            - type: integer
78
            - $ref: "../core/constant-1.0.0"
79
80
    allOf:
81
      - $ref: "transform-1.0.0"
82
83
      - properties:
          n_inputs:
84
            description: |
85
               Explicitly set the number of input axes. If not provided,
86
              it is determined from the maximum index value in the
87
               mapping list.
            type: integer
89
          mapping:
90
            $ref: "#/definitions/mapping"
91
        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:

```
0Type: transform-1.0.0 (page 58).1Type: 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
```

```
id: "http://stsci.edu/schemas/asdf/transform/add-1.0.0"
    tag: "tag:stsci.edu:asdf/transform/add-1.0.0"
5
    title: >
6
      Perform a list of subtransforms in parallel and then
      add their results together.
    description: |
10
      Each of the subtransforms must have the same number of inputs and
11
      outputs.
12
13
    examples:
14
15
        - A list of transforms, performed in parallel and added together
16
17
          !transform/add-1.0.0
18
            forward:
19
              - !transform/generic-1.0.0
20
                n_inputs: 1
21
                n_outputs: 2
22
              - !transform/generic-1.0.0
23
                n_inputs: 1
24
                n_outputs: 2
25
26
    allOf:
27
      - $ref: "transform-1.0.0"
28
29
      - properties:
          forward:
30
            type: array
31
            items:
32
              $ref: "transform-1.0.0"
33
        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:

```
0Type: transform-1.0.0 (page 58).1Type: 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
2
    $schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
3
    id: "http://stsci.edu/schemas/asdf/transform/subtract-1.0.0"
    tag: "tag:stsci.edu:asdf/transform/subtract-1.0.0"
5
    title: >
      Perform a list of subtransforms in parallel and then
      subtract their results.
    description: |
10
      Each of the subtransforms must have the same number of inputs and
11
12
13
      Invertibility: This transform is not automatically invertible.
14
    examples:
16
        - A list of transforms, performed in parallel, and then combined
17
          through subtraction.
18
19
          !transform/subtract-1.0.0
20
            forward:
21
              - !transform/generic-1.0.0
22
               n_inputs: 1
23
                n_outputs: 2
24
               - !transform/generic-1.0.0
25
                n_inputs: 1
26
                n_outputs: 2
27
28
    allOf:
29
      - $ref: "transform-1.0.0"
30
      - properties:
31
          forward:
32
            type: array
33
            items:
34
              $ref: "transform-1.0.0"
35
        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:

```
0Type: transform-1.0.0 (page 58).1Type: 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
2
    $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: |
10
     Each of the subtransforms must have the same number of inputs and
11
12
     Invertibility: This transform is not automatically invertible.
14
    examples:
15
16
        - A list of transforms, performed in parallel, and then combined
17
          through multiplication.
18
19
          !transform/multiply-1.0.0
20
21
              - !transform/generic-1.0.0
22
                n_inputs: 1
23
                n_outputs: 2
24
              - !transform/generic-1.0.0
25
                n_inputs: 1
27
                n_outputs: 2
28
    allOf:
29
      - $ref: "transform-1.0.0"
30
      - properties:
31
          forward:
32
            type: array
33
            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:

```
0Type: transform-1.0.0 (page 58).1Type: 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
```

```
%YAML 1.1
2
   $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: |
10
     Each of the subtransforms must have the same number of inputs and
11
12
13
     Invertibility: This transform is not automatically invertible.
   examples:
15
16
        - A list of transforms, performed in parallel, and then combined
17
         through division.
18
19
          !transform/divide-1.0.0
```

```
forward:
21
               - !transform/generic-1.0.0
22
                n_inputs: 1
23
                n_outputs: 2
24
               - !transform/generic-1.0.0
25
                n_inputs: 1
                n_outputs: 2
27
28
29
      - $ref: "transform-1.0.0"
30
      - properties:
31
32
          forward:
             type: array
33
             items:
34
               $ref: "transform-1.0.0"
35
        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:

```
0Type: transform-1.0.0 (page 58).1Type: any.
```

Original schema in YAML

```
%YAML 1.1
2
   $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: >
6
     Perform a list of subtransforms in parallel and then raise each
     result to the power of the next.
8
Q
10
   description: |
     Each of the subtransforms must have the same number of inputs and
11
     outputs.
12
13
     Invertibility: This transform is not automatically invertible.
14
15
   all0f:
16
     - $ref: "transform-1.0.0"
17
      - properties:
18
          forward:
```

```
type: array
items:

required: [forward]

type: array
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
2
   $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.
10
   type: object
11
   properties:
     offset:
13
        type: number
14
        description: Offset in one direction.
15
   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 1.1
2
   $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: >
6
     A Scale model.
   description: >
     Multiply the input by a factor.
10
   type: object
11
   properties:
12
     factor:
13
        type: number
14
        description: Multiplication factor.
15
   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:

```
Type: transform-1.0.0 (page 58).

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 $(n \times n)$, 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.
```

```
%YAML 1.1
2
    $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: |
8
     Invertibility: All ASDF tools are required to be able to compute the
9
      analytic inverse of this transform.
10
11
12
    all0f:
13
      - $ref: "transform-1.0.0"
14
      - type: object
15
        properties:
16
          matrix:
17
            description: |
              An array of size (*n* x *n*), where *n* is the number of axes,
19
              representing the linear transformation in an affine transform.
20
            anyOf:
21
              - $ref: "../core/ndarray-1.0.0"
22
              - type: array
23
                items:
                   type: array
25
                   items:
26
                    type: number
27
                  minItems: 2
28
                  maxItems: 2
29
                minItems: 2
30
31
                maxItems: 2
          translation:
32
            description: |
33
              An array of size (*n*,), where *n* is the number of axes,
34
              representing the translation in an affine transform.
```

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
2
   $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.
10
     Invertibility: All ASDF tools are required to be able to compute the
11
     analytic inverse of this transform.
12
    all0f:
13
      - $ref: "transform-1.0.0"
14
      - type: object
15
        properties:
16
          angle:
17
            type: number
            description: Angle, in degrees.
19
        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).
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.
```

Examples:

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

Default: "native2celestial"

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

```
%YAML 1.1
1
2
    $schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
3
    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.
9
10
      Invertibility: All ASDF tools are required to be able to compute the
11
      analytic inverse of this transform.
12
13
    examples:
14
15
        - The three Euler angles are 12.3, 34 and -1.2 in degrees.
16
17
          !transform/rotate3d-1.0.0
18
           phi: 12.3
19
            theta: 34
20
            psi: -1.2
21
            direction: zxz
22
23
    allOf:
24
      - $ref: "transform-1.0.0"
25
      - type: object
26
        properties:
27
          phi:
28
            type: number
29
            description: Angle, in degrees.
30
31
            type: number
32
            description: Angle, in degrees.
33
          psi:
35
            type: number
            description: Angle, in degrees.
36
          direction:
37
            description: |
38
              Sequence of rotation axes: one of 'zxz', 'zyz', 'yzy', 'yxy', 'xyx', 'xzx'
39
              or `native2celestial`, `celestial2native`.
40
41
              If 'direction' is 'native2celestial' or 'celestial2native',
42
              'phi', 'theta' are the longitude and latitude of the native pole in
43
              the celestial system and `psi` is the longitude of the celestial pole in
44
              the native system.
45
            enum: [zxz, zyz, yzy, yxy, xyx, xzx, native2celestial, celestial2native]
48
            default: native2celestial
49
        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:

$$\phi = \arg(-y, x)$$

$$R_{\theta} = \sqrt{x^2 + y^2}$$

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

$$x = R_{\theta} \sin \phi$$
$$y = R_{\theta} \cos \phi$$

```
All of:
```

```
Type: transform-1.0.0 (page 58).

Type: object.
Properties:
    direction
    Type: any from ["pix2sky", "sky2pix"].
    Default: "pix2sky"
```

```
%YAML 1.1
2
    $schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
3
    id: "http://stsci.edu/schemas/asdf/transform/zenithal-1.0.0"
     Base class of all zenithal (or azimuthal) projections.
6
    description: |
      Zenithal projections are completely specified by defining the radius
      as a function of native latitude, $R_\theta$.
10
11
      The pixel-to-sky transformation is defined as:
12
13
      $$\phi &= \arg(-y, x) \\
14
      R_{\text{theta \&= }} x^2 + y^2}
15
16
      and the inverse (sky-to-pixel) is defined as:
17
18
      x &= R_\theta \in \mathbb{N}
19
      y \&= R_\theta \sim \
20
21
    all0f:
22
      - $ref: "transform-1.0.0"
23
      - type: object
        properties:
25
          direction:
26
            enum: [pix2sky, sky2pix]
27
            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 1.1
2
    $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.
10
11
12
      [zenithal](ref:http://stsci.edu/schemas/asdf/transform/zenithal-1.0.0)
13
      for the definition of the full transformation.
14
15
      The pixel-to-sky transformation is defined as:
17
      \frac{180^{-1}\left(\frac{180^{\circ}}{\pi c}\right)}{\pi R_{\pi}}
18
19
      And the sky-to-pixel transformation is defined as:
20
21
      R_\theta = \frac{180^{\circ}}{\pi} \cot \theta
22
23
      Invertibility: All ASDF tools are required to provide the inverse of
24
      this transform.
25
26
    $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

```
%YAML 1.1
2
3 $schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
4 id: "http://stsci.edu/schemas/asdf/transform/zenithal_perspective-1.0.0"
```

```
tag: "tag:stsci.edu:asdf/transform/zenithal_perspective-1.0.0"
   title: |
6
     The zenithal perspective projection.
   description: |
     Corresponds to the 'AZP' projection in the FITS WCS standard.
10
11
     The pixel-to-sky transformation is defined as:
12
13
     $$\phi &= \arg(-y \cos \gamma, x) \\
14
     \theta &= \left\{\genfrac{}{}\phi}\\psi - \omega\{\psi + \omega + 180^{\circ}\\right.$$
15
16
17
     where:
18
     $$\psi &= \arg(\rho, 1) \\
19
     20
     \rho &= \frac{R}{\frac{180^{\circ}}{\min + 1} + y \sin \gamma} 
21
     R &= \sqrt{x^2 + y^2 \cos^2 \gamma}
22
23
     And the sky-to-pixel transformation is defined as:
24
25
     $$x &= R \sin \phi \\
26
     y \&= -R \sec \gamma \times \frac{1}{2}
2.7
28
     where:
29
30
     R = \frac{180^{\circ}}{\circ} \
31
32
     Invertibility: All ASDF tools are required to provide the inverse of
33
     this transform.
34
35
   all0f:
36
     - $ref: "zenithal-1.0.0"
37
     - type: object
38
       properties:
39
         mu:
40
           type: number
41
42
           description: |
            Distance from point of projection to center of sphere in
43
            spherical radii.
44
           default: 0
45
46
         gamma:
47
           type: number
48
           description: |
49
            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:
```

```
0Type: zenithal-1.0.0 (page 81).1Type: 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

```
%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:
16
17
      \frac{180^{-1}\left(\frac{180^{\circ}}{\pi R_\left(\frac{1}{\pi R}\right)}}{\pi R_\left(\frac{1}{\pi R}\right)}
18
19
      And the sky-to-pixel transformation is defined as:
20
21
      R_\theta = \frac{180^{\circ}}{\pi} \cot \theta
22
23
      Invertibility: All ASDF tools are required to provide the inverse of
24
      this transform.
25
26
    allOf:
27
      - $ref: "zenithal-1.0.0"
28
      - type: object
29
         properties:
30
           mu:
31
             type: number
32
             description: |
33
               Distance from point of projection to center of sphere in
               spherical radii.
35
             default: 0
36
37
           phi0:
38
             type: number
39
             description: |
               The longitude $\phi_0$ of the reference point, in degrees.
41
             default: 0
42
43
           theta0:
44
             type: number
45
             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 1.1
2
    $schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
3
    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.
10
11
12
      [zenithal](ref:http://stsci.edu/schemas/asdf/transform/zenithal-1.0.0)
13
      for the definition of the full transformation.
14
15
      The pixel-to-sky transformation is defined as:
16
17
      t = 90^{\circ} - 2 \tan^{-1}\left(\frac{pi R_{\phi}^{\circ}}{360^{\circ}}\right)
18
19
      And the sky-to-pixel transformation is defined as:
20
21
      R_\star = \frac{180^{\circ}}{\pi^2 2 \cos \theta} 1 + \sin \theta
22
23
      Invertibility: All ASDF tools are required to provide the inverse of
24
      this transform.
25
    $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.

```
The slant orthographic projection.
    description: |
9
      Corresponds to the 'SIN' projection in the FITS WCS standard.
10
11
12
      [zenithal](ref:http://stsci.edu/schemas/asdf/transform/zenithal-1.0.0)
13
      for the definition of the full transformation.
14
15
      The pixel-to-sky transformation is defined as:
16
17
      t = \cos^{-1}\left(\frac{\pi c^{\pi i}}{180^{circ}}R_{theta}\right)
18
19
      And the sky-to-pixel transformation is defined as:
20
21
      R_\theta = \frac{180^{\circ}}{\pi} \cos \theta
22
23
      Invertibility: All ASDF tools are required to provide the inverse of
24
      this transform.
25
    $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 1.1
2
3 $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.
```

```
15
      The pixel-to-sky transformation is defined as:
16
17
      $$\theta = 90^\circ - R_\theta
18
19
      And the sky-to-pixel transformation is defined as:
20
21
      R_\pm = 90^\circ - \theta
22
23
      Invertibility: All ASDF tools are required to provide the inverse of
24
      this transform.
25
    $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:

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

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

```
%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.
8
    description: |
     Corresponds to the 'ZEA' projection in the FITS WCS standard.
10
11
12
      [zenithal](ref:http://stsci.edu/schemas/asdf/transform/zenithal-1.0.0)
13
      for the definition of the full transformation.
15
      The pixel-to-sky transformation is defined as:
16
17
      t = 90\circ - 2 \sin^{-1} \left(\frac{\pi _{\pi _{\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:

Original schema in YAML

```
%YAML 1.1
2
   $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.
10
11
12
     [zenithal](ref:http://stsci.edu/schemas/asdf/transform/zenithal-1.0.0)
13
     for the definition of the full transformation.
14
15
   allOf:
16
     - $ref: "zenithal-1.0.0"
17
     - 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"
```

```
%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.
10
11
    all0f:
      - $ref: "transform-1.0.0"
12
      - type: object
13
        properties:
14
          direction:
15
            enum: [pix2sky, sky2pix]
16
            default: pix2sky
17
```

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 1.1
2 ---
3 $schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
4 id: "http://stsci.edu/schemas/asdf/transform/cylindrical_perspective-1.0.0"
5 tag: "tag:stsci.edu:asdf/transform/cylindrical_perspective-1.0.0"
6 title: |
7 The cylindrical perspective projection.
8
9 description: |
10 Corresponds to the `CYP` projection in the FITS WCS standard.
```

```
11
     The pixel-to-sky transformation is defined as:
12
13
     \phi \ \phi &= \frac{x}{\lambda} \\
14
     15
     And the sky-to-pixel transformation is defined as:
17
18
     $$x &= \lambda \phi \\
19
     20
21
     Invertibility: All ASDF tools are required to provide the inverse of
22
     this transform.
23
25
   allOf:
     - $ref: "cylindrical-1.0.0"
26
     - type: object
27
      properties:
28
29
        mu:
          type: number
30
         description: |
31
           Distance from center of sphere in the direction opposite the
32
           projected surface, in spherical radii.
33
         default: 0
34
35
        lambda:
37
          type: number
38
          description: |
           Radius of the cylinder in spherical radii, default is 0.
39
         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:

```
0Type: cylindrical-1.0.0 (page 92).
```

```
Type: object.
```

Properties:

lambda

Type: number.

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

Default: 0

Original schema in YAML

```
%YAML 1.1
2
   $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"
6
     The cylindrical equal area projection.
   description: |
     Corresponds to the `CEA` projection in the FITS WCS standard.
10
11
     The pixel-to-sky transformation is defined as:
12
13
     $$\phi &= x \\
14
     15
16
     And the sky-to-pixel transformation is defined as:
17
18
     $$x &= \phi \\
19
     y &= \frac{180^{\circ}}{\pi}\frac{180^{\circ}}{\pi}
20
21
     Invertibility: All ASDF tools are required to provide the inverse of
22
     this transform.
23
   allOf:
25
     - $ref: "cylindrical-1.0.0"
26
     - type: object
2.7
       properties:
28
         lambda:
29
           type: number
30
           description: |
31
             Radius of the cylinder in spherical radii, default is 0.
32
           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 1.1
2
    $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.
9
    description: |
     Corresponds to the `CAR` projection in the FITS WCS standard.
10
11
      The main virtue of this transformation is its simplicity.
12
13
      The pixel-to-sky transformation is defined as:
14
15
      $$\phi &= x \\
16
17
      \theta &= y$$
18
      And the sky-to-pixel transformation is defined as:
19
20
      $$x &= \phi \\
21
      y &= \theta$$
22
23
      Invertibility: All ASDF tools are required to provide the inverse of
      this transform.
25
26
    $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:

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

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 1.1
2
    $schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
3
    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.
10
11
      The pixel-to-sky transformation is defined as:
12
13
      $$\phi &= x \\
14
      \theta &= 2 \tan^{-1}\left(e^{y \pi / 180^{\circ}\right)-90^{\circ}$$
16
      And the sky-to-pixel transformation is defined as:
17
18
      $$x &= \phi \\
19
      y &= \frac{180^{\circ}}{\pi \left(\frac{90^{\circ}} + \frac{2}\right)}
20
21
      Invertibility: All ASDF tools are required to provide the inverse of
22
      this transform.
23
24
    $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:

```
0Type: transform-1.0.0 (page 58).1Type: 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"
3
   id: "http://stsci.edu/schemas/asdf/transform/pseudocylindrical-1.0.0"
   title: |
     Base class of all pseudocylindrical projections.
   description: |
8
     Pseudocylindrical projections are like cylindrical projections
9
     except the parallels of latitude are projected at diminishing
10
     lengths toward the polar regions in order to reduce lateral
11
     distortion there. Consequently, the meridians are curved.
12
13
   allOf:
14
      - $ref: "transform-1.0.0"
15
      - type: object
16
       properties:
17
          direction:
18
            enum: [pix2sky, sky2pix]
19
            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.

```
The Sanson-Flamsteed projection.
8
    description: |
9
      Corresponds to the 'SFL' projection in the FITS WCS standard.
10
11
      The pixel-to-sky transformation is defined as:
12
13
      $$\phi &= \frac{x}{\cos y} \\
14
        \theta &= y$$
15
16
      And the sky-to-pixel transformation is defined as:
17
18
      x = \phi \ \cos \theta \
19
        y &= \theta$$
20
21
      Invertibility: All ASDF tools are required to provide the inverse of
22
      this transform.
23
24
    $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
---
sschema: "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:
  12
 13
                                                  \hat x = \frac{180^\circ {180^\circ 
  14
                                                                \theta &= 3 \sin^{-1}\left(\frac{y}{180^\circ}\right)$
  15
                                                And the sky-to-pixel transformation is defined as:
  18
                                                  x &= \phi \left(2 \cos \frac{2\theta}{3} - 1\right) \
  19
                                                                y \&= 180^\circ \  \
 20
21
                                                Invertibility: All ASDF tools are required to provide the inverse of
22
                                                this transform.
 23
24
                                $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.

```
%YAML 1.1
---
sschema: "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}} \\
    \therefore \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \therefore
    \
```

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

$$x &= \frac{2 \sqrt{2}}{\pi (180^\circ \pi^2)} \cdot \frac{\pi^2}{\pi^2} \cdot
```

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.
10
11
      The pixel-to-sky transformation is defined as:
12
13
      \ \phi &= 2 \arg \left(2Z^2 - 1, \frac{\pi}{180^\circ} \frac{Z}{2}x\right) \\
14
        \theta = \sin^{-1}\left(\frac{\pi c^{\pi i}}{180^{circ}}z\right)
15
```

```
And the sky-to-pixel transformation is defined as:
17
18
     x &= 2 \gamma \ \cos \theta \ \sinh{2} \
19
      y &= \gamma \sin \theta$$
20
21
22
     where:
23
     s^{2}_1 + \cos \theta / 2}
24
25
     Invertibility: All ASDF tools are required to provide the inverse of
26
     this transform.
27
   $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:

$$\phi = \arg\left(\frac{Y_0 - y}{R_\theta}, \frac{x}{R_\theta}\right) / C$$

$$R_\theta = \operatorname{sign}\theta_a \sqrt{x^2 + (Y_0 - y)^2}$$

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 \text{ where } \theta_1 \text{ and } \theta_2 \text{ are the latitudes of the standard parallels, in degrees.} Default: 0 delta \text{Type: number.} (\theta_1-\theta_2)/2 \text{ where } \theta_1 \text{ and } \theta_2 \text{ are the latitudes of the standard parallels, in degrees.} Default: 0
```

Original schema in YAML

```
%YAML 1.1
1
2
   $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: |
8
     In conic projections, the sphere is thought to be projected onto the
9
     surface of a cone which is then opened out.
10
11
     In a general sense, the pixel-to-sky transformation is defined as:
12
13
      14
       R_\theta = \mathrm{sign} \cdot \mathrm{sign} \cdot \mathrm{syrt}(x^2 + (Y_0 - y)^2)
15
16
     and the inverse (sky-to-pixel) is defined as:
17
18
19
     x &= R_\theta \
       y &= R_\times \ (C \phi) + Y_0
20
21
     where $C$ is the "constant of the cone":
22
23
     SC = \frac{180^\circ \cos \theta}{pi R_\theta}
24
25
   allOf:
26
     - $ref: "transform-1.0.0"
27
      type: object
28
       properties:
29
         direction:
30
31
           enum: [pix2sky, sky2pix]
32
           default: pix2sky
33
         sigma:
34
           type: number
35
           description: |
36
             (\theta_1 + \theta_2) / 2 where \theta_1 = 1 and \theta_2
37
             are the latitudes of the standard parallels, in degrees.
38
           default: 0
39
40
         delta:
41
           type: number
42
           description: |
43
             (\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:

$$\begin{split} C &= \sin \theta_a \\ R_\theta &= \frac{180^\circ}{\pi} \cos \eta [\cot \theta_a - \tan (\theta - \theta_a)] \\ Y_0 &= \frac{180^\circ}{\pi} \cos \eta \cot \theta_a \end{split}$$

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_perspective-1.0.0"
    tag: "tag:stsci.edu:asdf/transform/conic_perspective-1.0.0"
6
     Colles' conic perspecitve projection.
   description: |
Q
     Corresponds to the `COP` projection in the FITS WCS standard.
10
11
12
      [conic](ref:http://stsci.edu/schemas/asdf/transform/conic-1.0.0)
13
      for the definition of the full transformation.
14
15
      The transformation is defined as:
16
17
18
      $$C &= \sin \theta_a \\
       R_{\pm \infty} = \frac{180\circ}{\pi - \tanh(-1.60\%)} \c
19
        Y_0 &= \frac{180^\circ \text{pi} \cos \beta \cdot \text{theta_a}}{
20
21
      Invertibility: All ASDF tools are required to provide the inverse of
22
     this transform.
23
    $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:

$$C = \frac{180^{\circ}}{\pi} \frac{\sin \theta_a \sin \eta}{\eta}$$

$$R_{\theta} = \theta_a - \theta + \eta \cot \eta \cot \theta_a$$

$$Y_0 = \eta \cot \eta \cot \theta_a$$

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

Original schema in 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: |
9
     Corresponds to the 'COD' projection in the FITS WCS standard.
10
11
12
     [conic](ref:http://stsci.edu/schemas/asdf/transform/conic-1.0.0)
13
     for the definition of the full transformation.
14
15
     The transformation is defined as:
16
17
     18
       R_\theta = \theta - \theta - \theta 
19
       Y_0 = \epsilon_\infty 
20
21
     Invertibility: All ASDF tools are required to provide the inverse of
22
     this transform.
23
24
   $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:

$$\begin{split} C &= \gamma/2 \\ R_{\theta} &= \frac{180^{\circ}}{\pi} \frac{2}{\gamma} \sqrt{1 + \sin \theta_1 \sin \theta_2 - \gamma \sin \theta} \\ Y_0 &= \frac{180^{\circ}}{\pi} \frac{2}{\gamma} \sqrt{1 + \sin \theta_1 \sin \theta_2 - \gamma \sin((\theta_1 + \theta_2)/2)} \end{split}$$

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
   2
                     $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.
 10
11
 12
                                [conic](ref:http://stsci.edu/schemas/asdf/transform/conic-1.0.0)
 13
                                for the definition of the full transformation.
 14
 15
                                The transformation is defined as:
 16
 17
                                $$C &= \gamma / 2 \\
18
                                         19
                                          Y_0 &= \frac{180^\circ (\theta_1 + \sin \theta_1 + \sin \theta_2 - \gamma_1 + \cot \theta_1 + \cot \theta_2 - \gamma_2 + \cot \theta_2 + \cot \theta_2
20
21
                                where:
22
23
                                \frac{1}{\sqrt{1 + \sin \theta}} 
24
25
                                Invertibility: All ASDF tools are required to provide the inverse of
26
                                this transform.
27
28
                     $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
2
   $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"
     Conic orthomorphic projection.
8
   description: |
9
     Corresponds to the 'COO' projection in the FITS WCS standard.
10
11
12
     [conic](ref:http://stsci.edu/schemas/asdf/transform/conic-1.0.0)
13
     for the definition of the full transformation.
14
15
     The transformation is defined as:
16
17
     $$C &= \frac{\ln \left( \frac{\cos\theta_2}{\cos\theta_1} \right)}
18
                 {\ln \left[ \frac{\tan\left(\frac{90^\circ-\theta_2}{2}\right)}
19
                                 {\time {\time {00^\circ - theta_1}{2}\rightarrow } \ } \ 
20
       21
       Y_0 = \phi \left( \frac{90^\circ - \theta^2}{1 - \theta^2} \right) \right)
22
23
     where:
25
     $$\psi = \frac{180^\circ}{\pi} \frac{\cos \theta}
26
              {C\left[\tan\left(\frac{90^\circ-\theta}{2}\right)\right]^C}$$
27
28
     Invertibility: All ASDF tools are required to provide the inverse of
29
     this transform.
31
   $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).
```

```
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.
10
11
   allOf:
      - $ref: "transform-1.0.0"
12
      - type: object
13
        properties:
14
          direction:
15
            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:

$$\begin{split} R_{\theta} &= \mathrm{sign}\theta_1 \sqrt{x^2 + (Y_0 - y)^2} \\ A_{\phi} &= \mathrm{arg}\bigg(\frac{Y_0 - y}{R_{\theta}}, \frac{x}{R_{\theta}}\bigg) \end{split}$$

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

$$x = R_{\theta} \sin A_{\phi}$$
$$y = -R_{\theta} \cos A_{\phi} + Y_{0}$$

where:

$$\begin{split} A_{\phi} &= \frac{180^{\circ}}{\pi R_{\theta}} \phi \cos \theta \\ R_{\theta} &= Y_0 - \theta \\ Y_0 &= \frac{180^{\circ}}{\pi} \cot \theta_1 + \theta_1 \end{split}$$

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.
```

Original schema in YAML

Default: 0

```
%YAML 1.1
2
    $schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
3
    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.
7
    description: |
10
      Corresponds to the 'BON' projection in the FITS WCS standard.
11
      The pixel-to-sky transformation is defined as:
12
13
      \ \phi &= \frac{\pi}{180^\circ R_\theta R_\theta / \cos \theta }
14
        \theta = Y_0 - R_\theta
15
16
      where:
17
18
      R_\theta = \mathrm{sign} \cdot \mathrm{sgrt}(x^2 + (Y_0 - y)^2) 
19
        A_\phi &= \arg\left(\frac{Y_0 - y{R_\theta}, \frac{x{R_\theta}\right)$$
20
21
      And the sky-to-pixel transformation is defined as:
22
23
      x = R_\theta \le R_\theta \
       y &= -R_{\text{theta }} cos A_{\text{phi}} + Y_0$
25
26
      where:
27
28
      A_\phi \ = \frac{180^\circ}{\pi R_\theta} \
29
        R_{\text{theta \&= Y_0 - \theta \ }}
```

```
Y_0 &= \frac{180^\circ \text{ } \cdot \text{ } \cdot
31
32
                                                                                            Invertibility: All ASDF tools are required to provide the inverse of
33
                                                                                            this transform.
  34
  35
                                                             allOf:
                                                                                         - $ref: "pseudoconic-1.0.0"
37
                                                                                            - type: object
38
                                                                                                                           properties:
39
                                                                                                                                                       theta1:
     40
                                                                                                                                                                                      type: number
     41
     42
                                                                                                                                                                                      description: |
                                                                                                                                                                                                                     Bonne conformal latitude, in degrees.
     43
                                                                                                                                                                                         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 1.1
2
   $schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
3
   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.
10
11
     Invertibility: All ASDF tools are required to provide the inverse of
12
     this transform.
13
14
   $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:

```
### All of:

### All of:

### Type: transform-1.0.0 (page 58).

### Type: object.

### Properties:

### direction

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

Default: "pix2sky"

```
%YAML 1.1
2
    $schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
    id: "http://stsci.edu/schemas/asdf/transform/quadcube-1.0.0"
     Base class of all quadcube projections.
    description: |
8
     Quadrilateralized spherical cube (quad-cube) projections belong to
9
      the class of polyhedral projections in which the sphere is projected
10
11
      onto the surface of an enclosing polyhedron.
12
      The six faces of the quad-cube projections are numbered and laid out
13
      as:
14
15
16
17
                  0
            4 3 2 1 4 3 2
18
                  5
19
20
21
    allOf:
22
     - $ref: "transform-1.0.0"
23
      - type: object
        properties:
25
          direction:
26
            enum: [pix2sky, sky2pix]
27
            default: pix2sky
28
```

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 1.1
2
   $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: |
6
     Tangential spherical cube projection.
   description: |
10
     Corresponds to the 'TSC' projection in the FITS WCS standard.
11
     Invertibility: All ASDF tools are required to provide the inverse of
12
     this transform.
13
14
   $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

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.

```
%YAML 1.1
2
   $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"
     Quadrilateralized spherical cube projection.
   description: |
9
     Corresponds to the 'QSC' projection in the FITS WCS standard.
10
11
      Invertibility: All ASDF tools are required to provide the inverse of
12
     this transform.
13
14
   $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:

```
O
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
```

```
%YAML 1.1
2
    $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.
8
    description: |
9
      Corresponds to the `XPH` projection in the FITS WCS standard.
10
11
      Invertibility: All ASDF tools are required to provide the inverse of
12
      this transform.
13
14
    allOf:
15
      - $ref: "transform-1.0.0"
16
      - type: object
17
        properties:
18
          direction:
19
            enum: [pix2sky, sky2pix]
20
            default: pix2sky
21
22
          H:
23
            type: number
24
            description: |
25
              The number of facets in the longitude direction.
26
            default: 4.0
27
28
          Χ:
29
            type: number
            description: |
31
              The number of facets in the latitude direction.
32
            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 1.1
2
    $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.
8
    description: |
      Corresponds to the `XPH` projection in the FITS WCS standard.
10
11
      Invertibility: All ASDF tools are required to provide the inverse of
12
      this transform.
13
14
    all0f:
15
      - $ref: "transform-1.0.0"
16
      - type: object
17
        properties:
18
19
          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
```

Items:

Type: ndarray-1.0.0 (page 20) or array. Required.

An array with coefficients.

Any of:

y of:
--Type: ndarray-1.0.0 (page 20).
--Type: array.

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}:
```

Original schema in YAML

```
%YAML 1.1
2
    $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: >
6
     A Polynomial model.
    description: |
     A polynomial model represented by its coefficients stored in
10
11
      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
12
      of the polynomial.
13
14
      p = \sum_{i, j=0}^{i+j=n}c_{ij} * x^{i} * y^{j}
15
16
      Invertibility: This transform is not automatically invertible.
17
18
    examples:
19
20
        - P = 1.2 + 0.3 * x + 56.1 * x^{2}
21
22
23
            !transform/polynomial-1.0.0
              coefficients: !core/ndarray-1.0.0
24
                               [1.2, 0.3, 56.1]
25
26
        - P = 1.2 + 0.3 * x + 3 * x * y + 2.1 * y^{2}
27
28
            !transform/polynomial-1.0.0
              coefficients: !core/ndarray-1.0.0
30
                               [[1.2, 0.0, 2.1],
31
                                [0.3, 3.0, 0.0],
32
                                [0.0, 0.0, 0.0]]
33
34
    type: object
35
    properties:
37
     coefficients:
        description: |
38
          An array with 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).
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.
```

Examples:

Create a regions selector schema for 2 regions, labeled "1" and "2".:

Value to be returned if there's no transform defined for the inputs.

```
!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
        [[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
          - !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
```

```
%YAML 1.1
2
    $schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
3
    id: "http://stsci.edu/schemas/asdf/transform/regions_selector-1.0.0"
    tag: "tag:stsci.edu:asdf/transform/regions_selector-1.0.0"
    title: >
6
     Represents a discontinuous transform.
    description: |
8
      Maps regions to transgorms and evaluates the transforms with the corresponding inputs.
9
10
11
    examples:
12
        - Create a regions_selector schema for 2 regions, labeled "1" and "2".
14
            !transform/regions_selector-1.0.0
15
              inputs: [x, y]
16
              label_mapper: !transform/label_mapper-1.0.0
17
                mapper: !core/ndarray-1.0.0
18
                   datatype: int8
19
                   data:
20
                     [[0, 1, 1, 0, 2, 0],
21
                      [0, 1, 1, 0, 2, 0],
22
                      [0, 1, 1, 0, 2, 0],
23
                      [0, 1, 1, 0, 2, 0],
24
                      [0, 1, 1, 0, 2, 0]]
25
              outputs: [ra, dec, lam]
27
              selector:
28
                1: !transform/compose-1.0.0
29
                   forward:
30
                   - !transform/remap_axes-1.0.0
31
                     mapping: [0, 1, 1]
32
                   - !transform/concatenate-1.0.0
33
                     forward:
34
                     - !transform/concatenate-1.0.0
35
                       forward:
36
                       - !transform/shift-1.0.0 {offset: 1.0}
37
                       - !transform/shift-1.0.0 {offset: 2.0}
38
                     - !transform/shift-1.0.0 {offset: 3.0}
                2: !transform/compose-1.0.0
40
                   forward:
41
                   - !transform/remap_axes-1.0.0
42
                     mapping: [0, 1, 1]
43
                   - !transform/concatenate-1.0.0
44
                     forward:
45
                     - !transform/concatenate-1.0.0
46
                       forward:
47
                       - !transform/scale-1.0.0 {factor: 2.0}
48
                       - !transform/scale-1.0.0 {factor: 3.0}
49
                     - !transform/scale-1.0.0 {factor: 3.0}
50
              undefined_transform_value: .nan
51
52
53
    allOf:
54
      - $ref: "transform-1.0.0"
55
      - type: object
```

```
properties:
57
          label_mapper:
58
             description: |
59
               An instance of
60
               [label_mapper-1.0.0](ref:http://stsci.edu/schemas/asdf/transform/label_mapper-1.0.0)
61
             $ref: "./label_mapper-1.0.0"
62
           inputs:
63
             description: |
64
               Names of inputs.
65
             type: array
66
             items:
67
               type: string
69
          outputs:
             description: |
70
               Names of outputs.
71
             type: array
72
             items:
73
               type: string
74
           selector:
             description: |
76
               A mapping of regions to trransforms.
77
             type: object
78
             properties:
79
               labels:
80
                 description: |
                   An array of unique region labels.
                 type: array
83
                 items:
84
                   type:
85
                      - integer
86
                     - string
87
               transforms:
88
                 description: |
89
                   A transform for each region. The order should match the order of labels.
90
                 type: array
91
                 items:
92
                   $ref: "transform-1.0.0"
93
          undefined_transform_value:
95
             description: |
               Value to be returned if there's no transform defined for the inputs.
96
             type: number
97
        required: [label_mapper, inputs, outputs, selector]
98
```

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)).
                 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:
```

5.5. Transform

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
```

```
%YAML 1.1
2
    $schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
3
    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
10
      [regions_selector](ref:http://stsci.edu/schemas/asdf/transform/regions_selector-1.0.0). The
11
      [label_mapper](ref:http://stsci.edu/schemas/asdf/transform/label_mapper-1.0.0)
12
      returns the label corresponding to given inputs. The
      [regions_selector](ref:http://stsci.edu/schemas/asdf/transform/regions_selector-1.0.0)
14
      returns the transform corresponding to this label. This maps inputs
15
      (e.g. pixels on a detector) to transforms uniquely.
16
17
    examples:
18
19
        - Map array indices are to labels.
20
21
22
            !transform/label_mapper-1.0.0
23
              mapper: !core/ndarray-1.0.0
24
                [[1, 0, 2],
25
                [1, 0, 2],
                [1, 0, 2]]
27
28
29
        - Map numbers dictionary to transforms which return labels.
30
31
32
            !transform/label_mapper-1.0.0
33
              mapper: !!omap
34
              - !!omap
35
                labels: [-1.67833272, -1.9580548, -1.118888]
36
              - !!omap
37
                models:
38
              - !transform/compose-1.0.0
                forward:
40
                - !transform/remap_axes-1.0.0
41
                  mapping: [1]
42
                - !transform/shift-1.0.0 {offset: 6.0}
43
              - !transform/compose-1.0.0
                forward:
45
                - !transform/remap_axes-1.0.0
                  mapping: [1]
```

```
- !transform/shift-1.0.0 {offset: 2.0}
48
               - !transform/compose-1.0.0
49
                 forward:
50
                 - !transform/remap_axes-1.0.0
51
                   mapping: [1]
52
                 - !transform/shift-1.0.0 {offset: 4.0}
53
               inputs: [x, y]
54
               inputs_mapping: !transform/remap_axes-1.0.0
55
                 mapping: [0]
56
                 n_inputs: 2
57
58
59
         - Map a number wihtin a range of numbers to transforms which return labels.
60
61
62
             !transform/label_mapper-1.0.0
63
               mapper: !!omap
64
               - !!omap
65
                 labels:
                 - [3.2, 4.1]
67
                 - [2.67, 2.98]
68
                 - [1.95, 2.3]
69
               - !!omap
70
                 models:
71
               - !transform/compose-1.0.0
72
                 forward:
73
                 - !transform/remap_axes-1.0.0
74
                   mapping: [1]
75
                 - !transform/shift-1.0.0 {offset: 6.0}
76
               - !transform/compose-1.0.0
77
                 forward:
78
                 - !transform/remap_axes-1.0.0
                   mapping: [1]
80
                 - !transform/shift-1.0.0 {offset: 2.0}
81
               - !transform/compose-1.0.0
82
                 forward:
83
                 - !transform/remap_axes-1.0.0
84
                   mapping: [1]
85
                 - !transform/shift-1.0.0 {offset: 4.0}
               inputs: [x, y]
87
               inputs_mapping: !transform/remap_axes-1.0.0
88
                 mapping: [0]
89
                 n_inputs: 2
90
91
    allOf:
92
      - $ref: "transform-1.0.0"
93
      - type: object
94
         properties:
95
           mapper:
96
             description: |
97
               An array with the shape of the detector/observation.
98
               Pixel values are of type integer or string and represent
               region labels.
100
               Pixels which are not within any region have value 0 or " ".
101
             anyOf:
102
               - $ref: "../core/ndarray-1.0.0"
103
               - type: object
104
                 properties:
```

```
labels:
106
                        type: array
107
                        items:
108
                          anyOf:
109
                             - type: number
110
                             - type: array
111
                               items:
112
                                 type: number
113
                               minLength: 2
114
                               maxLength: 2
115
                     models:
116
117
                        type: array
118
                        items:
                          $ref: "transform-1.0.0"
119
120
            inputs:
121
              type: array
122
              items:
123
124
                type: string
            inputs_mapping:
125
              $ref: "transform-1.0.0"
126
127
          required: [mapper]
128
```

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).
```

```
%YAML 1.1
2
    $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
9
     detector space to world coordinate space and back) by using the
10
     transformations in the `transform-schema` module.
11
12
    type: object
13
    properties:
      name:
14
        description: |
15
          A descriptive name for this WCS.
16
        type: string
17
18
      steps:
19
        description: |
20
          A list of steps in the forward transformation from detector to
21
          world coordinates.
22
          The inverse transformation is determined automatically by
23
          reversing this list, and inverting each of the individual
24
          transforms according to the rules described in
25
          [inverse] (ref: http://stsci.edu/schemas/asdf/transform/transform-1.0.0/properties/inverse).\\
26
        type: array
27
        items:
28
          $ref: step-1.0.0
29
30
    required: [name, steps]
31
    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 1.1
2
    $schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
3
    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: []
10
    type: object
11
    properties:
12
      frame:
13
        description: |
          The frame of the inputs to the transform.
        anyOf:
16
          - type: string
17
          - $ref: frame-1.0.0
18
19
      transform:
20
        description: |
21
          The transform from this step to the next one. The
22
          last step in a WCS should not have a transform, but
23
          exists only to describe the frames and units of the
24
          final output axes.
25
        anyOf:
26
          - $ref: ../transform/transform-1.0.0
27
          - type: 'null'
        default: null
29
30
    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.h

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).
```

5.6. WCS 129

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
```

The distance from the sun to the galactic midplane. Required when reference_frame is galactocentric. Required when reference_frame is galactocentric.

Items:

Type: array.

```
index[0]
  Type: number.
  index[1]
  Type: unit-1.0.0 (page 46).
  Default: "pc"
roll
```

The angle to rotate about the final x-axis, relative to the orientation for galactic. Required when reference_frame is galactocentric.

Items:

Type: array.

```
index[0]
Type: number.
index[1]
Type: unit-1.0.0 (page 46).
Default: "deg"
obsgeoloc
```

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:

Type: array.

```
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]
```

```
%YAML 1.1
2
    $schema: "http://stsci.edu/schemas/yaml-schema/draft-01"
3
    id: "http://stsci.edu/schemas/asdf/wcs/frame-1.0.0"
    title: |
      The base class of all coordinate frames.
    description: |
8
      These objects are designed to be nested in arbitrary ways to build up
9
      transformation pipelines out of a number of low-level pieces.
10
11
      Most of these coordinate frames are defined in [IERS
12
      conventions](http://www.iers.org/IERS/EN/Publications/TechnicalNotes/tn36.html).
13
14
    examples:
15
16
17
            A celestial frame in the FK4 reference frame.
18
19
            !wcs/celestial_frame-1.0.0
20
              axes_names: [ra, dec]
21
              name: CelestialFrame
22
              reference_frame:
23
                type: FK4
24
                equinox: !time/time-1.0.0 '2010-01-01 00:00:00.000'
25
                obstime: !time/time-1.0.0 '2015-01-01 00:00:00.000'
26
              unit: [!unit/unit-1.0.0 deg, !unit/unit-1.0.0 deg]
27
28
    type: object
29
    properties:
30
      name:
31
        description: |
32
          A user-friendly name for the frame.
33
        type: string
34
35
      axes_order:
36
        description: |
37
          The order of the axes.
38
        type: array
39
        items:
40
          type: integer
41
42
      axes_names:
43
        description: |
44
45
          The name of each axis in this frame.
        type: array
        items:
          anyOf:
48
            - type: string
49
            - type: 'null'
50
51
      reference_frame:
52
53
        description: |
          The reference frame.
54
        type: object
55
        properties:
```

```
57
           type:
             description: |
58
               The reference frame type. Some reference frame types
59
               require additional properties, listed next to each reference
60
               frame type below.
61
62
               The reference frames types are:
63
               - `ICRS`
65
66
               - `FK5`: `equinox`.
67
68
               - `FK4`: `equinox` and optionally `obstime`.
69
70
               - `FK4_noeterms`: `equinox` and optionally `obstime`.
71
72
               - `galactic`
73
74
               - `galactocentric`: `galcen_distance`, `galcen_ra`,
75
                 `galcen_dec`, `z_sun` and `roll`.
76
77
               - `GCRS`: `obstime`, `obsgeoloc`, and `obsgeovel`.
78
79
               - `CIRS`: `obstime`.
80
81
               - `ITRS`: `obstime`.
82
83
               - `precessed_geocentric`: `obstime`, `obsgeoloc`, and
84
                 `obsgeovel`.
85
86
             enum: [ICRS, FK5, FK4, FK4_noeterms, galactic, galactocentric,
87
                    GCRS, CIRS, ITRS, precessed_geocentric]
88
             default: ICRS
89
90
           equinox:
             description: |
92
               The equinox of the reference frame. Required when
93
               `reference_frame` one of:
94
95
                 `FK5`, `FK4`, `FK4_noeterms`
96
97
             $ref: ../time/time-1.0.0
98
99
           obstime:
100
             description: |
101
               The observation time of the reference frame, used to determine
102
               the location of the Earth. Required when `reference_frame` is
103
               one of:
104
105
                 `FK4`, `FK4_noeterms`, `GCRS`, `CIRS`, `ITRS`
106
107
               If not provided, it defaults to the same value as 'equinox'.
108
             $ref: ../time/time-1.0.0
109
110
           galcen_distance:
111
             description: |
112
               The distance from the Sun to the Galactic center. Required when
113
               `reference_frame` is `galactocentric`.
114
```

```
type: array
             items:
116
                - type: number
117
                - $ref: ../unit/unit-1.0.0
118
                 default: pc
119
120
           galcen_ra:
121
             description: |
               The Right Ascension (RA) of the Galactic center in the ICRS
123
               frame. Required when `reference_frame` is `galactocentric`.
124
             type: array
125
             items:
126
               - type: number
127
               - $ref: ../unit/unit-1.0.0
128
                 default: deg
129
130
           galcen_dec:
131
             description: |
132
               The Declination (DEC) of the Galactic center in the ICRS frame.
133
               Required when `reference_frame` is `galactocentric`.
134
135
             type: array
             items:
136
               - type: number
137
               - $ref: ../unit/unit-1.0.0
138
                 default: deg
139
140
           z_sun:
             description: |
142
               The distance from the sun to the galactic midplane. Required
143
               when `reference_frame` is `galactocentric`. Required when
144
                `reference_frame` is `galactocentric`.
145
             type: array
146
             items:
147
               - type: number
149
               - $ref: ../unit/unit-1.0.0
                 default: pc
150
151
           roll:
152
             description: |
153
               The angle to rotate about the final x-axis, relative to the
               orientation for `galactic`. Required when `reference_frame` is
155
                `galactocentric`.
156
             type: array
157
             items:
158
               - type: number
159
               - $ref: ../unit/unit-1.0.0
160
161
                 default: deg
162
           obsgeoloc:
163
             description: |
164
               3-vector giving the position of the observer relative to the
165
               center-of-mass of the Earth, oriented the same as
166
               BCRS/ICRS. Defaults to `[0, 0, 0]`, meaning "true" GCRS. Used
167
               when `reference_frame` is `GCRS` or `precessed_geocentric`.
168
             type: array
169
             items:
170
                - type: array
171
                 items:
172
```

```
type: number
173
                 minItems: 3
174
                 maxItems: 3
175
               - $ref: ../unit/unit-1.0.0
176
                 default: m
177
             default:
               - [0, 0, 0]
179
180
           obsgeovel:
181
             description: |
182
               3-vector giving the velocity of the observer relative to the
183
               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`.
186
             type: array
187
             items:
188
               - type: array
189
                 items:
190
                    type: number
191
                 minItems: 3
192
                 maxItems: 3
193
               - $ref: ../unit/unit-1.0.0
194
                 default: m/s
195
             default:
196
               - [0, 0, 0]
         required: [type]
199
200
      unit:
201
         description: |
202
           Units for each axis.
203
         type: array
204
         items:
205
           $ref: ../unit/unit-1.0.0
206
207
    required: [name]
208
    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.

Type: frame-1.0.0 (page 127).
```

```
%YAML 1.1
2
    $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.
10
11
     type: object
        properties:
12
          axes_names:
13
            minItems: 2
14
            maxItems: 3
15
16
17
          axes_order:
18
            minItems: 2
            maxItems: 3
19
20
          unit:
21
            minItems: 2
22
            maxItems: 3
23
      - $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).
```

```
%YAML 1.1
2
    $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
10
        properties:
11
          reference_position:
12
13
            description: |
              The position of the reference frame.
14
            enum: [geocenter, barycenter, heliocenter]
15
            default: geocenter
16
17
          axes_names:
18
            minItems: 1
            maxItems: 1
20
21
          axes_order:
22
            minItems: 1
23
            maxItems: 1
24
25
          unit:
            minItems: 1
27
            maxItems: 1
28
29
      - $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.
```

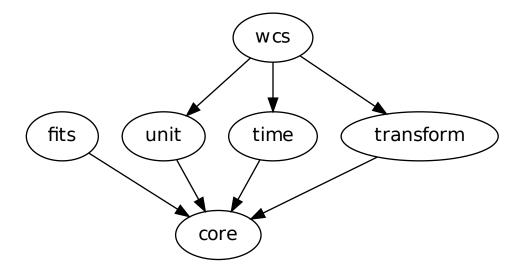
Properties:

```
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).
```

```
%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
10
        properties:
11
          name:
12
            description:
13
              Name of composite frame.
14
            type: string
15
          frames:
17
            description:
18
              List of frames in the composite frame.
19
            type: array
20
21
      - $ref: frame-1.0.0
```

The following graph shows the dependencies between modules:



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).
```

Type: object.

Properties:

tag

Type: string ($len \ge 6$).

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 1.1
1
2
    $schema: "http://json-schema.org/draft-04/schema"
3
    id: "http://stsci.edu/schemas/yaml-schema/draft-01"
    title:
     YAML Schema
    description: |
     A metaschema extending JSON Schema's metaschema to add support for
8
     some YAML-specific constructions.
9
    allOf:
10
      - $ref: "http://json-schema.org/draft-04/schema"
11
      - type: object
12
        properties:
13
          tag:
14
            description: |
15
              A fully-qualified YAML tag name that should be associated
16
              with the object type returned by the YAML parser; for
17
              example, the object must be an instance of the class
18
              registered with the parser to create instances of objects
19
20
              with this tag. Implementation of this validator is optional
              and depends on details of the YAML parser.
21
            type: string
22
            minLength: 6
23
24
          propertyOrder:
25
            description: |
              Specifies the default order of the properties when writing
27
              out. Any keys not listed in propertyOrder will be in
28
              arbitrary order at the end.
29
            type: array
30
            items:
31
32
              type: string
33
          flowStyle:
34
            description: |
35
              Specifies the default serialization style to use for an
36
              array or object. YAML supports multiple styles for
37
              arrays/sequences and objects/maps, called "block style" and
38
              "flow style". For example::
39
40
                Block style: !!map
41
                  Clark : Evans
42
                  Ingy : döt Net
43
                  Oren : Ben-Kiki
44
45
                Flow style: !!map { Clark: Evans, Ingy: döt Net, Oren: Ben-Kiki }
              This property gives a hint to the tool outputting the YAML
48
              which style to use. If not provided, the library is free to
49
              use whatever heuristics it wishes to determine the output
50
              style. This property does not enforce any particular style
51
              on YAML being parsed.
52
            type: string
53
            enum: [block, flow]
54
55
          style:
56
```

6.1. YAML Schema 143

```
description: |
57
               Specifies the default serialization style to use for a string.
58
               YAML supports multiple styles for strings::
59
60
                 Inline style: "First line\nSecond line"
61
62
                 Literal style: |
63
                   First line
64
                   Second line
65
66
                 Folded style: >
67
                   First
                   line
69
70
                   Second
71
                   line
72
73
               This property gives a hint to the tool outputting the YAML
74
               which style to use. If not provided, the library is free to
75
               use whatever heuristics it wishes to determine the output
76
               style. This property does not enforce any particular style
77
               on YAML being parsed.
78
             type: string
79
             enum: [inline, literal, folded]
80
81
82
          examples:
            description: |
83
               A list of examples to help document the schema. Each pair
84
               is a prose description followed by a string containing YAML
85
               content.
86
             type: array
87
             items:
88
               type: array
89
               items:
90
                 - type: string
91
                 - type: string
92
```

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
```

6.1. YAML Schema

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:

 ${\tt max_ndim}$

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 1.1
2
    $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:
10
      - $ref: "http://stsci.edu/schemas/yaml-schema/draft-01"
11
      - type: object
12
        properties:
13
          max_ndim:
            description: |
15
              Specifies that the corresponding ndarray is at most the
16
              given number of dimensions. If the array has fewer
17
              dimensions, it should be logically treated as if it were
18
              "broadcast" to the expected dimensions by adding 1's to the
19
              front of the shape list.
20
            type: integer
21
            minimum: 0
22
23
          ndim:
24
            description: |
25
              Specifies that the matching ndarray is exactly the given
26
              number of dimensions.
            type: integer
28
            minimum: 0
29
30
          datatype:
31
            description: |
32
              Specifies the datatype of the ndarray.
33
34
              By default, an array is considered "matching" if the array
35
              can be cast to the given datatype without data loss. For
36
              exact datatype matching, set `exact_datatype` to `true`.
37
            allOf:
38
              - $ref: "http://stsci.edu/schemas/asdf/core/ndarray-1.0.0#definitions/datatype-1.0.0"
39
          exact_datatype:
41
42
            description: |
              If `true`, the datatype must match exactly.
43
            type: boolean
44
            default: false
```

6.2. ASDF Schema 147

46 ...

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, F00, 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:F00/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/F00/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:F00/metadata-1.0.0"
id: "http://stsci.edu/schemas/F00/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 F00 instrument.
description: |
   This stores some information about an exposure from the F00 instrument.
examples:
   -
    - A minimal description of an exposure.
   - |
        !F00/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:F00/metadata-1.0.0"
id: "http://stsci.edu/schemas/F00/metadata-1.0.0"

title: |
   Metadata for the F00 instrument.
description: |
   This stores some information about an exposure from the F00 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_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:
required: [exposure_time]
additionalProperties: true
```

6.4 Extending an existing schema

TODO

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.

CHANGES

8.1 Version 1.0.0

First pre-release.

156 Chapter 8. Changes

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.

```
|BITPIX = -64|
|NAXIS| = 2
|NAXIS1| = 512
|NAXIS2| = 512
|EXTNAME = 'DQ '
|END
|XTENSION= 'IMAGE '
|BITPIX = 8
|NAXIS| = 1
|NAXIS1 = 361
|EXTNAME = 'ASDF '
IEND
|#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

[Thomas 2015] 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.