Is my data file PDS4-compliant?

PDS4 data structures are severely limited, and how they may be applied to data is also part of PDS4 compliance. PDS4 data structures are designed for long-term archive stability and to support interdisciplinary use. Specifically, for archival science data the primary data structures are tables and arrays of 2-4 dimensions. These simple structures not only ensure long-term stability in the archive, but they are also relatively hard to mis-read, reducing the amount of end-user error resulting from misunderstood record formats. The use of these simple, standardized formats also supports inter-disciplinary use of the data, as it tends to severely limit dependence on specific software packages for working with PDS4 data.

Choosing Data Structures

You should choose PDS4 data structures that match the logical view of your data, as broken down into a series of tables and arrays. For example, a simple image is archived as a 2D array object, *not* as a table of vectors. Similarly, if you have a series of 2D arrays containing ancillary data for an image - like quality flags, dark current, bad pixel maps, and such - each of those 2D arrays must be archived as a separate data structure, *not* as pseudo-bands in an image cube.

File Structure

Multiple data objects may be stored in a single data file, and frequently are. All PDS4 data structures stored in a file must be distinct from each other and contiguous in themselves. So you may not, for example, interleave records of a table with scan lines of an image - you must separate the table out into one contiguous block of bytes and the image into another. The table and image may, of course, be stored sequentially in the same data file once separated into distinct data objects.

Non-PDS Data Formats

Sometimes it happens that data formatted to some other sort of standard (a processing standard or a transport standard) may, coincidentally, already exist in a file as a series of separate arrays and tables that are consistent with the logical view of the data, and thus meet the PDS4 data structure requirements. In these cases, the PDS4 label may be written to describe the data as it exists in the original file. The PDS4 *Header* object may be used to indicate a header block formatted according to another standard that is also included in the file (a VICAR or FITS header, for example). But even when the data structure in the file is PDS4-compliant, the PDS4 label must contain all the information needed for a user to read and interpret the data. The *Header* object is merely a way of accounting for non-PDS4-compliant bytes; end users must be able to ignore it without consequence.

> **Note:** Even though some files formatted to other data standards may be PDS4 compliant, there is no guarantee that any other file written in that standard will be. *There is no other data format standard known as of this writing that is guaranteed to produce a PDS4-compliant data file in all cases.* Every science product archived with the PDS *must* be in a PDS4-compliant format. It is the data preparer's responsibility to ensure PDS4 compliance for all products submitted for archiving.

FITS Format

FITS is a transport format common in small bodies data. Many FITS files contain data structures that are PDS4-compliant, though some data preparers use the format in non-standard ways or pack multiple, logically distinct arrays into a single multi-dimensional IMAGE in a way that is not PDS4-compliant.

How do I translate FITS Header values into PDS label values?

FITS is a transport format originally designed for moving observational data between different computer architectures. Many FITS files are, structurally at least, compliant with PDS4 data structure requirements, but care must be taken to ensure that the FITS structures are not being abused, or that logically distinct data objects are not being packed into a single data structure, which would negate PDS4 compliance.

Follow the guidelines below for filling in the PDS4 attribute values for compliant FITS files.

FITS Data Structures

Many FITS files are also PDS4-compliant, but no FITS structure is *guaranteed* to be PDS4-compliant. Following are the issues to be aware of for the potentially PDS4-compliant FITS files.

BINTABLE

FITS **BINTABLE** extension data is structurally compliant with PDS4 binary table requirements. It is possible to abuse this format by using the binary table record options in FITS to include a multi-dimensional array in the table record. If this is done to accommodate, for example, a vector field or a small matrix of values, then this can be considered PDS4-compliant and labelled using *Group_Field_Binary* structures in the PDS4 label for the vector and array data. If this is being used to attach header values to an observational data array, however, it violates the physical vs. logical relationship required for PDS4 science data. Such data would generally need to be reformatted.

IMAGE

The **IMAGE** extension, as well as the primary data segment, contains an N-dimensional array. Any array with less than two, or more than four dimensions is suspect.

Degenerate 1D arrays have sometimes been used rather than **BINTABLE** extensions to avoid the more detailed description requirements. Often this can be addressed simply by using the appropriate binary table description in the PDS4 label without reformatting the data itself (the appropriateness of the FITS description is not of primary concern for PDS4 archiving, although the external peer reviewers may insist that it be corrected or removed if they consider it misleading).

Higher-dimensional arrays have been used to contain inhomogeneous data in a single array - for example, a 2D intensity image might be combined with a quality map, bad pixel map, and temperature map - and labelled in the FITS file as a 3D image. This is not PDS4 compliant. Each plane with a different interpretation *must* be tagged as a separate data object in a PDS4 label (and similarly for higher-dimensional cases). This may or may not require reformatting the data, depending on how the inhomogeneous planes are stored.

Legitimate degenerate and higher-order arrays do sometimes occur. If you think you have such a case in hand, contact your PDS node consultant for advice on how to proceed.

TABLE

The **TABLE** extension indicates a fixed-width ASCII table. Some FITS ASCII tables are PDS4-compliant, but any candidate table should be examined carefully because the PDS4 constraints on ASCII tables are a little more restrictive than the FITS constraints.

As in PDS4, only 7-bit ASCII characters are permitted in FITS *TABLE*s, and the only non-printable characters permitted in either FITS or PDS4 character tables are the blank character and the carriage-return and linefeed characters, which may only be used as record delimiters. FITS, however, has no requirements on which line delimiters, if any, are used in the data. PDS4 requires that all lines end with first a carriage-return and then a linefeed. If the FITS table does not have the correct delimiters, it must be modified before it can be labelled as a PDS4 table.

Other FITS Structures

Note: Other FITS data structures, like random groups or ASCII tables without the PDS4-required line delimiters, are not PDS4-compliant and must be converted to a compliant format before they may be archived.

Special Note About Signed/Unsigned Integers

There seems to be an occasional misunderstanding about Table 19 in the FITS standard document that affects both arrays and binary tables.

The only integer data types stored in a compliant FITS file are unsigned 8-bit bytes and signed 2, 4, and 8-byte integers. Table 19 of the FITS standard does give the correct offset for converting each of these types to their signed and unsigned counterparts, respectively, *provided you have properly converted the numbers from one hardware storage type to the other*. It is **not true** that you can, for example, just store unconverted unsigned 2-byte integers in your FITS file, add the offset from Table 19, and expect to get the correct (unsigned) value back from a compliant FITS reader.

The table in the FITS standard assumes that before your write your FITS file you properly convert unsigned integers to signed integers by subtracting the offset value given and converting to a signed hardware data type ***before*** you write your data into the FITS file. If you have not done this, then the offset required to recover the original unsigned value is, in fact, *twice* the offset value shown in that table if the value read in is negative, and *zero* if it is positive. (And conversely going from signed to unsigned bytes.)

So, if you are migrating a PDS3-labeled FITS file and you see a *DATA_TYPE* keyword in the PDS3 label describing a one-byte field as an "MSB_INTEGER", or a multi-byte field as an "MSB_UNSIGNED_INTEGER", you should be extremely skeptical about that data type description in the PDS3 label, and possibly in the FITS header as well. Make sure that the offsets are consistent with the values expected and found in the file, and report any discrepancies you discover.

In all cases the PDS4 *<data_type>* ***must*** be the data type used to read the values in the file, and *not* the data type you might ultimately want to end up with. That conversion is a private matter between the end user and his software.

Discipline Dictionaries

If your FITS files contain image or spectral data, you will need to provide additional information to indicate the proper way to display them to correctly interpret things like axis labels and observation geometry. Typically, any label for image data will include a *<Display_Settings>* class from the *Display* discipline dictionary. See the Filling Out the Display Dictionary Class page for details. If your file also contains spectral data, you should also check the Filling Out the Spectral Dictionary Class page.

Depending on the data, there may be additional discipline dictionary classes required in your PDS4 label, as well as any mission dictionary classes that might be appropriate.

Headers

Use the PDS4 *Header* object to describe any type of FITS header.

- The <*object_length*> must be the number of FITS blocks comprising the header * 2880 (the size of a single FITS block).
- For <*parsing_standard_id*>, use "FITS 3.0".

Note that the FITS header is not considered an archival object by PDS4; the *Header* object merely provides enough information for a program to skip over the header. Any critical information in the FITS header *must* be translated into the PDS4 label.

Arrays

This category includes both the primary data array, following the first FITS header in the file, and the data in any *IMAGE* extension. Use <*Array_2D_Image*> for any 2D data that can be reasonably considered an image. For other data, use the most specific and relevant flavor of *Array_** available.

Axis Ordering

FITS array data are stored such that the *NAXIS1* subscript varies fastest, *NAXIS2* next fastest, and so on up to *NAXISn*. So, the storage order is first-index-fastest in FITS notation.

In PDS, arrays are stored so that axis 1 (the axis described by the <*Axis_Array*> that has <*sequence_number*> of "1") varies least rapidly, axis 2 next least rapidly, and so on to axis *n*.

So, when labeling a FITS array as a PDS *Array*-type object, the highest-numbered *NAXIS** becomes axis 1 in the PDS array, the next-highest *NAXIS** becomes axis 2, and so on.

Array Element Description

In the FITS primary data header or image extension header, the following reserved FITS keywords also have direct PDS4 equivalents in the <*Element_Array*> class.

FITS	**PDS4 *Array***
BSCALE	<Element_Array>/<scaling_factor>
BZERO	<Element_Array>/<value_offset>
BUNIT	<Element_Array>/<unit>
BLANK *integer data only*	*see below*
DATAMAX	<Object_Statistics>/<maximum_scaled_value>
DATAMIN	<Object_Statistics>/<minimum_scaled_value>

The *BLANK* keyword, when used according to the FITS standard, may only be used with integer data, and may either be a blank (if BSCALE is 1 and BZERO is 0), otherwise it contains a flag value indicating null data. If you need to specify a null value, add the <*Special_Constants*> class to the *Array* class, and select the appropriate special constant for the case in hand. Note that the *BLANK* value is the value read from the file - *before* any scale factor and offset are applied.

Scalar Data Types

In both the primary data array and *IMAGE* extensions, the data type is indicated by the value of the *BITPIX* keyword.

BITPIX Value	**PDS Data Type(s)**

8	UnsignedByte *-or-* 7-bit ASCII character
16	SignedMSB2 (integer)
32	SignedMSB4 (integer)
-32	IEEE754MSBSingle (float)
64	SignedMSB8 (integer)
-64	IEEE754MSBDouble (float)

2D Images

In addition to the above correspondences, the PDS4 *Array_2D_Image* class has the following additional correspondences.

- *NAXIS1* corresponds to axis 2, and axis 2 **must** have an <*axis_name*> of "Sample".
- *NAXIS2* corresponds to axis 1, and axis 1 **must** have an <*axis_name*> of "Line".

Also note that the FITS standard says nothing about display direction. It seems to be universally true that samples (NAXIS1) are always drawn left-to-right; but lines are drawn either top-to-bottom or bottom-to-top with roughly equal frequency. You will have to look at your FITS images and determine the correct order of display. You *must* indicate this order in your label via the **Display Discipline Dictionary** classes.

Binary Tables

Use a <*Table_Binary*> class for FITS *BINTAB* tables.

Note that the FITS standard requires that the BINTAB *data be padded with null ("\0") from the end of the last record to the end of the containing 2880-byte FITS block. There is no requirement to document these padding bytes in the PDS4 label, and they are generally ignored.*

Field Descriptions

In the FITS *BINTAB* extension, the field description keywords, for the most part, translate directly to PDS4 *Field_Binary* class attributes. Note that the only *required* keyword for each field is *TFORM*:

FITS	PDS4 *Field_Binary*
TTYPE	<name>
TUNIT	<unit>
TSCAL	<scaling_factor>
TZERO	<value_offset>
TNULL *integer data only*	*see below*
TDISP	*see below*
TFORM	<data_type> *see below*

TNULL is only properly used for integer data. It indicates a null data flag, and corresponds to the value found in the file - before scaling or offset are applied. It can be translated into one of the specific flags in the <*Special_Constants*> class, depending on the circumstances of the data.

TDISP provides a recommended print format for the binary data, using a limited set of FORTRAN-like specifiers. In PDS4 the specifiers follow the POSIX standard and are described on the PDS4 Field Format Conventions page.

Note: The *TDISP* value may include a repetition specifier (as the first number in the string). If this number is zero, the field is non-existent in the FITS file and should NOT have a corresponding *Field_Binary* in the PDS label. If this number is greater than one, then the field is either a vector or a multi-dimensional array. See the sections below on "Vector Fields" and "Array Fields" for more information.

TFORM is used in a *BINTAB* table to define the storage data type of the associated field. These and their PDS4 equivalents are enumerated below.

Note: In the *BINTAB* extension, there is no way to specify the location of a particular field as an offset from the beginning of the record, as the <*field_location*> attribute does for PDS4 files. This means two things:

1. Every byte in the *BINTAB* record *must* be included in one of the defined fields - there is no undeclared gutter space, spare bytes, or record delimiters in FITS *BINTAB* data. On top of that, the fields *must* be defined in the order in which they appear in the file.
2. You will have to calculate the <*field_location*> for each field in the record by adding up the total sizes of all preceeding fields. *Don't forget the repetition counts in your* TFORM *fields in your location math!*

Scalar Data Types

TFORM will have to be translated to the PDS4 equivalent. The FITS *BINTAB* scalar types that are directly supported by PDS4 binary tables are:

FITS Type Letter	PDS4 <*data_type*>
B	UnsignedByte
I	SignedMSB2
J	SignedMSB4
K	SignedMSB8
A	ASCII_String (*or any other appropriate* ASCII_* *type*)
E	IEEE754MSBSingle
D	IEEE754MSBDouble
C	ComplexMSB8
M	ComplexMSB16

"*Other appropriate* ASCII_* *types*" include things like *ASCII_Date_Time* for strings representing times in the standard format, for example.

In addition to the scalar types, the FITS standard allows a field to contain a repeating value (basically, a vector), fixed-sized array, or a variable-sized array descriptor.

Vector Fields

Vector fields can be identified in the FITS header by finding fields with a repetition count greater than one in their *TFORM* values, but without an accompanying *TDIM* value for the field (or a *TDIM* that has only one subscript).

Note: For ASCII types, and *only* ASCII types, a repeat count in the *TFORM* value should be treated as the length of the string (i.e., the size of the field), and *not* as indicating an array of single-character values. In the PDS4 label, use a *Group_Field_Binary* class to define FITS vector fields. The *Group_Field_Binary* will contain have a *<fields>* value of 1 and *<groups>* of 0, with a *<repetitions>* value equal to the repetition count from the *TFORM* value. Nested inside the group will be a single *<Field_Binary>* definition to define the repeating element.

Array Fields

There are two possible array types in FITS binary tables: Fixed-size arrays, which are supported in PDS4; and variable-size arrays, which are *not* PDS4-compliant.

Fixed-size arrays look like vector fields, in that they have *TFORM* values with a repetition count greater than one. But they also have an associate *TDIM* keyword with a value that looks like a set of array indices. For example, this set of keywords defines a 5x6 array of double-precision floating point numbers as a single FITS binary table field:

```
        TTYPE01 = 'SIMPLE ARRAY'
        TFORM01 = '30D'
        TDIM01  = '(5,6)'
```

The repetition count in *TFORM* ("30", in this case) *must* be equal to the total number of elements in the array.

N.B.: FITS stores *all* arrays, including binary table array fields, in first-index-fastest order. The PDS4 label must describe this array in a way that will ensure that it is read in that sequence. In order to do this, you will need nested *Group_Field_Binary* classes, with a *Field_Binary* class at the bottom. So, the basic structure for describing the field above would be:

```
        <Group_Field_Binary>
          <fields>0</fields>
          <groups>1</fields>
          <repetitions>6</repetitions>
          <group_location unit="byte">[whatever the start byte is for the whole group]</
      group_location>
          <group_length unit="byte">240</group_length>

          <Group_Field_Binary>
            <fields>1</fields>
            <groups>0</groups>
            <repetitions>5</repetitions>
            <group_location unit="byte">1</group_location>
            <group_length unit="byte">40</group_length>

            <Field_Binary>
              ...the usual field definition...
            </Field_Binary>
          </Group_Field_Binary>
        </Group_Field_Binary>
```

You should also add specific comments (in the <*description*> of the *Field_Binary*) to advise end-users on what each of the dimensions represented by the *Group_Field_Binary* classes represents, sufficient for a programmer to be able to decide the correct subscript order to use in mapping values from the data file into program memory.

ASCII Tables

FITS character tables will always appear as *TABLE* extensions. Only 7-bit ASCII characters are permitted in *TABLE*s, and the only non-printable character permitted in FITS or PDS4 character tables are the blank character and the carriage-return and linefeed characters, which may only be used as record delimiters.

Note that the FITS standard requires that the TABLE *data be padded with blanks to an even number of FITS 2880-byte blocks following the last record in the table. There is no requirement that these padding characters be included in the PDS4 label, and they're generally ignored.*

A Note About Line Delimiters

The FITS 3.0 standard allows non-printing control characters to appear at the end (after the last field) of each table record. Most commonly, any control characters found here will be some form of line delimiter. So, in FITS TABLE data:

- *Record delimiters are not required at all.*
- *Neither does the FITS standard require that all line delimiters, if they are present, be the same.*
- *If a delimiter* is *present, you have no standard way of determining what it is without actually reading a record and checking the bytes.*

PDS, on the other hand, requires that every line of *Table_Character* data end with a "carriage-return/linefeed" sequence. The upshot of all this is that you cannot assume that any FITS *TABLE* extension can automatically be labelled as a PDS4 *Table_Character*. You ***must*** ensure that carriage-return/linefeed line delimiters exist on every record before the data can be archived.

TABLE Header Keywords

In the *TABLE* extension header:

- *NAXIS1* corresponds to the <*record_length*> attribute in the <*Record_Character*> class.
- *NAXIS2* corresponds to the <*records*> attribute in the <*Table_Character*> class.
- *TFIELDS* corresponds to the <*fields*> attribute in the <*Record_Character*> class. (Unlike in *BINTAB* tables, ASCII tables may not have fields that are arrays, vectors, or complex numbers.)

The required <*groups*> attribute will always have a value of zero (fields in FITS *TABLE*s are not allowed to have repetition counters).

Field Descriptions

Each field must have a corresponding *TBCOL* and *TFORM* keyword in the FITS *TABLE* header, but all other keywords are optional. The reserved keywords used to define the fields have close analogs in the PDS4 <*Field_Character*> class:

FITS	PDS4 ***Field_Character***
TBCOL	<field_location>
TFORM	<data_type> *see following*
TTYPE	<name>
TUNIT	<unit>
TSCAL	<scaling_factor>
TZERO	<value_offset>
TNULL	*see below*
TDISP	*see below*

TNULL is a string used as a null data flag for the field and corresponds to the value found in the file - before scaling or offset are applied. It can be translated into one of the specific flags in the <*Special_Constants*> class, depending on the circumstances of the data.

TDISP is an optional keyword that can be used to contain a more specific display format than that implied by the *TFORM* keyword.

Scalar Data Types

TFORM is required for every field to indicate data type. It has a syntax similar to a FORTRAN format specifier, with a type specifier followed by a total field width ('w') and, for real values, a precision ('d'). Unlike the keyword of a similar name in BINTAB extensions, *TFORM* values in TABLE extensions ***may not*** contain repetition counts in a compliant FITS file. [*Note:* Non-compliant FITS files have been known...]

The correspondence to PDS4 <*data_type*> is straightforward for numeric types:

TFORM	PDS4 <data_type>
A*w*	ASCII_String (*or more specific* ASCII_* *string type*)
I*w*	ASCII_Integer
F*w.d*	ASCII_Real
E*w.d*	ASCII_Real
D*w.d*	ASCII_Real

"*More specific* ASCII_* *string types*" include, for example, ASCII_Date_Time for date/time fields conforming to the standard ISO 8601 format.