Planetary Data System Standards Reference

1 May 2013

Version 1.0.0

Contents

1 Introduction

Version 4 of the Planetary Data System (PDS) is now available. Data from new missions are being designed for ingestion into 'PDS4' and earlier PDS3 data sets are being 'migrated' to the new PDS4 system[1].

The PDS4 Information Model (IM) is the fundamental reference for PDS4 structure; its requirements can be validated automatically using eXtensible Markup Language (XML) schemas.

The PDS4 Data Dictionary Data Base (DDDB) is the fundamental reference for definitions of classes and attributes.

The *PDS4 Concepts* document provides an overview of the philosophy behind and organization of PDS4. It includes a glossary of terms as used within PDS4.

1.1 Purpose

The PDS4 *Standards Reference* (*SR*) is a compilation of policies, rules, and other PDS4 constraints that are not given explicitly in the IM and DDDB. The combination of the IM, DDDB, and SR gives the set of 'common' requirements that apply across PDS. Each PDS discipline nodes (DNs) may establish additional requirements that further constrain (but do not conflict with) the 'common' requirements and apply only to data ingested through that node.

1.2 Scope

The PDS4 *SR* applies to PDS Version 4 and its holdings. External standards, adopted by PDS, are included by reference. Examples and recommended best practices can be found in the PDS4 *Data Provider's Handbook* (*DPH*) and references therein.

1.3 Audience

The PDS4 *SR* is intended primarily to serve the community of scientists and engineers responsible for preparing planetary science data for submission to PDS4. Such submissions include restored data from the era prior to PDS or from earlier versions of PDS, mission data from active and future planetary missions, and data from Earth-based sites, including laboratories and independent research efforts. The audience includes personnel at PDS discipline and data

[1] PDS4 standards are not backwardly compatible with version 3 (PDS3). In principle, version 4 data can be described using version 3 labels, but the converse is not true; some PDS3 structures are no longer supported under PDS4.

nodes, principal investigators and their staffs, and ground data system engineers. The document will be most useful to people who have experience with PDS archiving; those new to PDS may find the *PDS4 Concepts* document to be a better place to start.

1.4 Document Organization

The PDS4 *SR* is divided into two main sections. The first, Sections 1-7, contains detailed information on the structural aspects of a PDS4 archive: the format of data, labels, and their assembly into 'packages' for submission and transfer. The second, Section 8, is more focused on the nature of *what* is stored in PDS4 archives — such as adequacy of calibration and documentation.

1.5 External Standards

External standards, which apply to this document and to PDS4-compliant data, include the following:

American National Standards Institute (ANSI)

- ANSI INCITS 4-1986 (R2007) *Information Systems - Coded Character Sets - 7-Bit American National Standard Code for Information Interchange (7-Bit ASCII)*

Consultative Committee for Space Data Systems (CCSDS)

- CCSDS 641.0-B-2 *Parameter Value Language Specification (CCSD0006 and CCSD0008)* (also available as ISO 14961:2002)

Institute of Electrical and Electronics Engineers (IEEE)

- IEEE 754-2008 *Standard for Binary Floating-Point Arithmetic*

International Standards Organization (ISO)

- ISO 646:1991 ISO *7-bit coded character set for information interchange*
- ISO 8601:2004 *Data Element and Interchange Formats – Representations of Dates and Times*
- International Standards Organization/International Electrotechnical Commission (ISO/IEC) 10646:2012 *Information technology – Universal Coded Character Set (UCS)*
- ISO/IEC 11179-3:2003 *Metadata registries (MDR) – Part 3: Registry metamodel and basic attributes*

- ISO/IEC 11404:2007 *General-Purpose Datatypes (GPD)*

- ISO 14721:2003 *Open archival information system – Reference model*

- International Standards Organization / Technical Standard (ISO/TS) 15000-3:2004 *Electronic business eXtensible Markup Language (ebXML) – Part 3: Registry information model specification (ebRIM)*

- ISO/TS 15000-4:2004 *ebXML – Part 4: Registry services specification (ebRS)*

National Institute of Standards and Technology (NIST)

- NIST Special Publication 330 *The International System of Units (SI)*, United States version of the English text of the eighth edition (2006), Issued March 2008

World Wide Web Consortium (W3C)

- *XML 1.1* 2nd edition, August 16, 2006

- *XML Schema Part 0: Primer* 2nd edition, October 28, 2004 (W3C, 2004a)

- *XML Schema Part 1: Structures* 2nd edition, October 28, 2004 (W3C, 2004b)

- *XML Schema Part 2: Datatypes* 2nd edition, October 28, 2004 (W3C, 2004c)

Requests for Comment

- RFC 1321, The MD5 Message-Digest Algorithm (http://tools.ietf.org/html/rfc1321)

International DOI Federation

- http://www.doi.org General information and "Resolve a DOI Name" interface

1.6 Document Availability

PDS4 documents governing archive preparation are available online at:

http://pds.nasa.gov/pds4/doc/

For questions concerning these documents, contact any PDS data engineer or contact the PDS Operator at pds_operator@jpl.nasa.gov or 818-393-7165.

Associated schemas for current and past versions of PDS4 can be found at

http://pds.nasa.gov/pds4/schema/released/

2 Archive Organization

2A Archive Logical Organization

2A.1 Bundles

A *bundle* is the default logical construct for archiving digital data in the PDS.

Bundles have a simple hierarchical structure. A bundle has one or more member *collections*, each of which has one or more member *basic products* (Figure 2A.1). PDS does not impose requirements on how bundles are defined except that (1) bundles must be distinct within the overall holdings of PDS, and (2) each bundle must be approved by a PDS peer review.

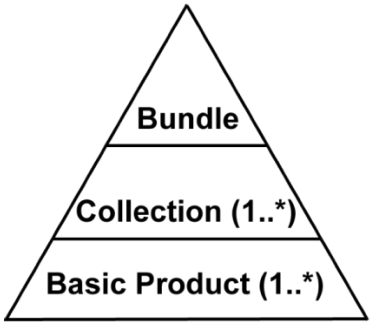


Figure 2A.1: Archive structure.

Members of a bundle are listed in a Product_Bundle, an XML file which serves as both a label and the bundle inventory. It is described using the Product_Bundle class definition (see *Data Provider's Handbook* for information on constructing Product_Bundle). Any single version of a bundle has one and only one version of Product_Bundle.

An optional "readme" file may be included as part of Product_Bundle; it is described by the bundle label so is not a separate product. It provides a general overview of the bundle contents and organization in human readable format. It may also contain general instructions for use of the bundle and contact information for data provider or discipline node personnel. The "readme" file may be formatted either as plain text or HTML.

2A.2 Collections

Data are organized into collections based on the type and function of the data. PDS imposes only broad requirements on how these type and function boundaries are drawn (see Section 2A.5). Collections must be distinct within a bundle, products must be distinct within each collection, and each collection must be approved by a PDS peer review. Any single version of a collection must have one and only one version of Product_Collection — an XML label file paired with an inventory table, which lists collection members for that version (see *Data Provider's Handbook* for information on constructing Product_Collection).

2A.3 Products

A *basic product* is the simplest product in PDS4 — one or more data objects and their description objects, which constitute a single observation, document, etc. Typically, a *data object* is a file containing a single image, table, or time series; *description objects* are typically text that describes both the format and content of the associated data object. A *label* is an XML file, which is the concatenation of one or more closely related *description objects* (such as for the red, green, and blue components of a color image) with some XML overhead; the corresponding basic product is that label plus the RGB data objects. A document basic product is constructed in the same way: the basic product is the set of files containing text, figures, and tables together with a label comprising the several description objects. Certain XML files qualify as basic products by themselves — they are 'XML documents'.

Digital objects which comprise observational data may be used in one and only one product.

Product_Collection and Product_Bundle are 'aggregate' products; they define an aggregation of basic products and an aggregation of collections, respectively. They are not basic products. All products, whether basic or aggregate, must have globally unique logical identifiers (see Section 6D).

2A.4 Primary and Secondary Members

Basic products may be either *primary* or *secondary* members of their respective collections. A primary member is one that is being registered with PDS for the first time. A secondary member is one which has already been registered with PDS, but which is now being associated with a different collection[2]. A product's member status (primary or secondary) is based on its *first* association with the collection. Although the product may be omitted from a later version of the collection, it retains its primary or secondary member status through all subsequent versions of the collection based on its initial association. In a similar way, collections are categorized as having either primary or secondary 'member status' in their bundles.

2A.5 Collection Types

Basic products must be organized into collections based on the type and function of the data. Data providers are expected to assign products to appropriate collections (*e.g.*, only 'quick-look' or 'browse' products are expected to be assigned to a *browse* collection); the assignments will be considered during peer review. PDS recognizes the following types of collections:

A *browse* collection contains 'quick-look' products designed to facilitate use of the archive. Products in a browse collection are defined using appropriate classes; possibilities include Product_Browse and Product_Thumbnail.

A *calibration* collection contains data and files necessary for the calibration of basic

[2] Data providers are not obligated to include physical copies of secondary member products or collections when they deliver bundles to PDS, since the secondary members already exist elsewhere within PDS. PDS, on the other hand, must deliver copies of these secondary members when bundles are distributed (unless waived by the recipient).

products. Products in a calibration collection are defined using the appropriate class; possibilities include Product_Observational and Product_Document.

A *context* collection is the list of products comprising various objects, identified within the PDS4 registry, that are specific to the science bundle. These include physical objects such as instruments, spacecraft, and planets and conceptual objects such as missions and PDS nodes. All products in a context collection will be secondary members of the collection, since the primary versions are curated by the PDS Engineering Node.

A *data* collection contains observational products, often separated according to processing level, target, instrument mode, etc.; these are the science data that most users seek. Products in a data collection will typically be defined using the Product_Observational class.

A *document* collection includes all components (figures, tables, text, etc.) of one or more closely related documents. The document collections included with a bundle are those deemed useful by the data preparer and consulting node for understanding, interpreting, and using other collections in the bundle. Documents may include software interface specifications (SISs), calibration reports, and data acquisition summaries, among other topics. Products in a document collection will be defined using the Product_Document class.

A *geometry* collection contains non-SPICE geometry products — for example, Supplementary Experiment Data Record (SEDR) data, gazetteers, tables of anaglyph pairs, or footprint files. Detailed information about particular cartographic projections utilized in the archive may also be included. Products in a geometry collection are defined using an appropriate class; possibilities include Product_Observational and Product_Document.

A *miscellaneous* collection contains supplementary information deemed by the data provider to be useful in the interpretation and use of other collections in the bundle but which does not fit within the scope of the other collections. For example, a table of updates to product label values made after the bundle has been ingested by PDS may be included in a miscellaneous collection. Other examples include meta-data catalogs, data base dumps, and records of modification history. Because of the disparate nature of the files included in a miscellaneous collection, no single class exists for miscellaneous products. The following are some of the possibilities: Product_Document, Product_File_Text, Product_Observational, Product_Thumbnail, Product_Update, and Product_Zipped.

An *XML schema* collection contains all XML schema files included in or referenced by XML labels in the bundle along with any Schematron files created for validation purposes. An XML catalog file may also be included. A bundle is required to contain an XML schema collection unless all of the collection members would be secondary in which case the XML schema collection becomes optional. There can be no more than one XML schema collection per bundle.

A *SPICE kernels* collection contains individual SPICE files and their XML labels, organized by kernel type. Products in a SPICE kernels collection must be defined using the Product_SPICE_Kernel class.

2B Archive Physical Organization

2B.1 PDS Data Storage and Access

Storage and accessibility of data within PDS must meet system requirements, which are addressed elsewhere. Except for file and directory naming (Section 6C) and the restrictions immediately below, no standards are set within this document for the organization of the physical archive (*i.e.*, there are no requirements for how the data are stored within or accessed from a physical repository or data storage).

2B.1.1 Objects and Files

PDS requires that each digital object be distinct and contiguous in a single file; objects may not overlap. For example, a color image may be stored as three separate Array_2D_Image (RGB) objects with each of the R, G, and B components being stored separately. The lines or samples of the three components may not be interleaved in this case. However, it is permissible to define such an image as a *single* Array_3D_Image object with interleaved lines or samples.

Although a single file may contain multiple digital objects, no digital object may extend beyond a single file. For example, the R, G, and B image components above may be stored (in that order) in a single file. Having any component (*e.g.*, R, G or B) start in one file and continue into a second file is not allowed.

2B.2 Data Transfer

The parties involved in a data transfer to, from, or within PDS are free to adopt any procedure that is mutually acceptable. In lieu of such an *ad hoc* agreement, the following will be used.

2B.2.1 Transfer Media

The parties involved in a data transfer must agree on the physical media and interfaces to be used (magnetic tape, CD, DVD, magnetic disk, electronic, etc.). PDS defers to industry standards for the media selected.

2B.2.2 Directory Structure

~~D~~ata will be transferred using a directory structure that *parallels* the logical organization of a corresponding bundle (Section 2A). Note that the directory structure does not itself have to meet the requirements for logical bundles or collections. Many transfers will contain only a portion of a bundle or collection.

The following sections describe such a directory structure, starting with the root directory. Under the root are subdirectories — *i.e.*, collections, following the logical organization of the bundle — that have controlled names. Under the controlled-name subdirectories are optional

sub-directories[3], for which there is no name control (except in the case of a SPICE kernels collection, where name control extends one more level; see Section 2B.2.2.3).

In the tables and figures below, the following conventions have been adopted:

- **Bold-faced** font is used for directory names, while unbolded font is used for file names. Where no pattern for directory or file name construction is specified, example names are shown in *italics*.

- The asterisk "*" is a 'wild card'. In Table 2B.1 **browse[_*]** means that "**browse**" or any directory name starting with "**browse_**" is acceptable. See Section 6C for restrictions on directory and file names.

- Square brackets "[ ]" identify variable parts of file or directory names. For example, readme[_*].[html,txt] in Table 2B.1 means that the file name beginning with "readme" may be lengthened with any character string that begins with an underscore "_". It also means that the file name extension (the character string following the final period ".") may be either "html" or "txt".

2B.2.2.1 Root Directory

The root directory corresponds to the top level of a bundle. It may contain a Product_Bundle and subdirectories corresponding to at least some of its member collections.

Table 2B.1 – *root* Directory

File or Directory Name	Notes
root	may be unnamed
bundle[_*].xml	
readme[_*].[html,txt]	described by bundle[_*].xml
browse[_*]	
calibration[_*]	
context[_*]	
data[_*]	
document[_*]	
geometry[_*]	
miscellaneous[_*]	
xml_schema[_*]	
spice_kernels[_*]	

A bundle is described by a label with the name "bundle[_*].xml". One or more of these files

[3] The hyphen in "sub-directories" distinguishes these lower levels from the top-level subdirectories (no hyphen), which correspond to collections.

may be included under the root of the transfer directory structure, but none is required.

The optional "readme" file provides a general overview of the bundle contents and organization in human readable format. The file is an optional component of Product_Bundle; it is not a separate product. Thus, it shares a label with the bundle. It is described using the File_Area_Text class within the Product_Bundle.

2B.2.2.2 Subdirectories

Beneath the ***root*** directory, the top-level subdirectories have a one-to-one correspondence with the bundle's collections. Within each collection subdirectory there will usually be an XML label with a name of the form "collection[_*].xml"; each label file is accompanied by a collection inventory table with a name of the form "collection[_*].tab". See the *Data Provider's Handbook* for details on construction of collection tables.

Labels for collections and the associated collection inventory tables must reside in the top-level subdirectories (*vid*., Table 2B.2). The member products for each of the collections must reside in lower level sub-directories (Figure 2B.1) An exception may be made for collections containing two dozen or fewer products, in which case the products and their labels may be co-located with the collection label in the top-level subdirectory (Figure 2B.2). The lower-level sub-directories are named and organized at the discretion of the data provider except for SPICE kernels sub-directories (see Section 2B.2.2.3).

Table 2B.2 – SPICE_Kernels Subdirectory

File or Directory Name	Notes
spice_kernels_*	
collection[_*].xml	
collection[_*].tab	
ck	as needed
dbk	as needed
dsk	as needed
ek	as needed
fk	as needed
ik	as needed
lsk	as needed
mk	as needed
pck	as needed
sclk	as needed
spk	as needed

In cases where a bundle contains multiple collections of the same type, the names of the subdirectories containing these collections must be distinguished by a suffix of the data provider's choosing appended to the mandated subdirectory prefix. For example, if two

calibration collections are present in the archive, the two calibration subdirectories might be named **calibration_*flight*** and **calibration_*ground*** (Figure 2B.3).

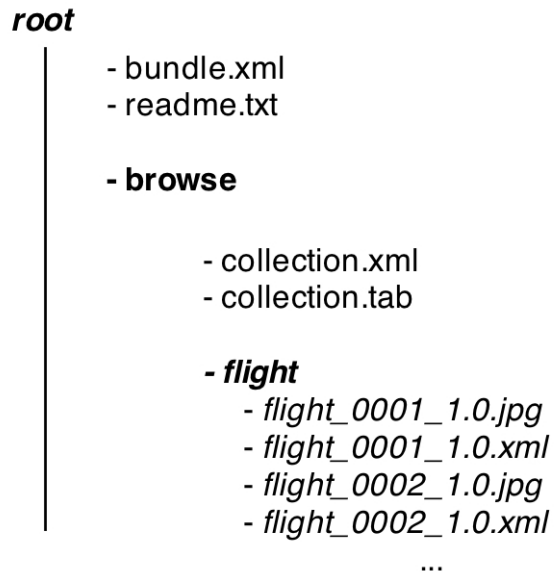


Figure 2B.1: Illustration of the structure for one **browse** subdirectory (one collection) with many basic products organized into sub-directories.

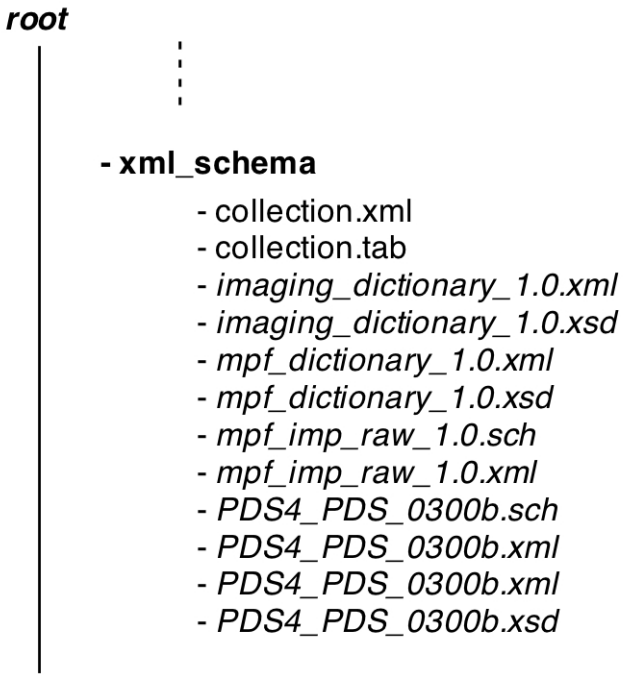


Figure 2B.2: Illustration of the structure for one **xml_schema** subdirectory (one collection) with a small number of basic products (*e.g.*, two dozen or fewer) placed at the same level as the collection label and inventory table.

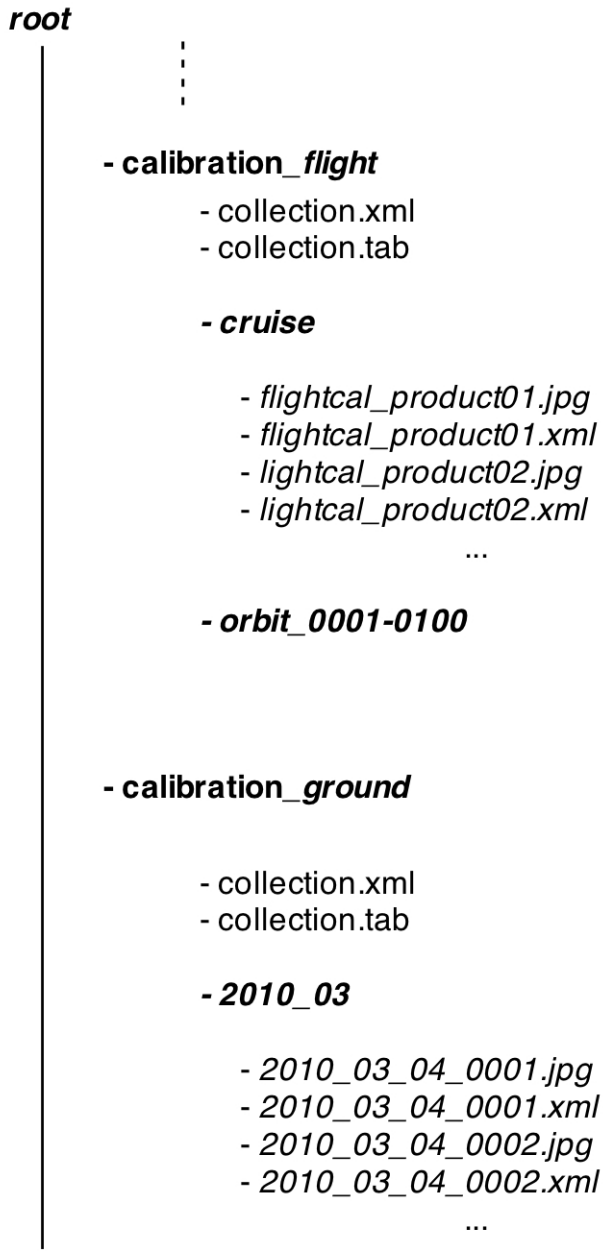


Figure 2B.3: Illustration of the structure for two subdirectories containing two collections of the same type (calibration). Lower level sub-directories (***cruise***, ***orbit_0001-0100***, ***2010_03***, and possibly others) are defined and named by the data provider.

2B.2.2.3 SPICE_Kernels Subdirectory and Sub-Directories

A SPICE_Kernels subdirectory contains individual SPICE files and their XML labels, organized by kernel type (see Table 2B.2), which must be placed in the corresponding sub-directories:

ck	CK files (spacecraft and instrument orientation data)
dbk	DBK files (databases in SPICE format)
dsk	DSK files (digital shape data for natural bodies)
ek	EK files (events information)
fk	FK files (reference frames definitions)
ik	IK files (instrument parameters and FOV definitions)
lsk	LSK files (leapsecond information)
mk	MK files (meta-kernels listing kernels to be used together)
pck	PCK files (natural body rotation and size/shape constants)
sclk	SCLK files (spacecraft clock correlation data)
spk	SPK files (trajectory and ephemeris data)

Note that SPICE kernel files have specific naming requirements. In particular, there are mandated file name extensions based on the type of kernel file (see Section 6C).

3 Labels

PDS product labels are required for describing the contents and format of each individual product within an archive. Specifically, the labels contain the description objects that describe corresponding data objects. The description objects are populated using a standard set of classes, attributes, and standard values, which are themselves defined in the PDS4 Information Model. The PDS4 Information Model is expressed, and therefore PDS labels are written, in the eXtensible Markup Language (XML). Thus a PDS label is an "XML document".

In order for any XML document (including a PDS label) to meet the XML standard, it must be both "well formed" and "valid". A well-formed XML document must have correct XML syntax; a valid XML document must conform to the rules of an XML schema document (XSD). XSDs provide the rules governing the structure and content of a specific XML document class. PDS splits these latter two functions between XSDs (structure) and Schematron files (content).

Thus, in order for a label to comply with PDS4 standards, it must:

- have correct XML syntax
- be compliant with the class and attribute structures defined by the PDS and relevant discipline nodes and missions in their respective XSDs
- be compliant with the rules governing specific attributes and their values as set by the PDS and relevant discipline nodes and missions in their respective Schematron files
- have a file name ending with the extension "xml".

PDS4 schemas are supplied to data providers by the PDS. Missions and other data providers must not modify these pre-existing schemas; however, they may extend existing classes and provide additional attributes in their own dictionary schemas, with the approval of a PDS discipline node.

Under PDS4, all product labels are detached from the files containing the digital objects they describe. There is one PDS4 label for every product. Each product may contain one or more data objects. The digital objects of a given product may all reside in a single file[4], or they may be stored in multiple, separate files. However, a single digital object may not reside in multiple files.

Figure 3-1 shows the general structure of a label, simplified to components that appear in many labels. "XML Declaration and Schema Reference" is a few lines of XML overhead required for label implementation; the remainder of the label is defined by the PDS4 Information Model. `Root Tag`, `File_Area_Definition`, and `End Tag` are placeholders, which vary among labels; the first points to the data object, the second describes format and content of the associated physical file(s) if present, while the third marks the end of the label. `Identification_Area` provides 'fingerprints' and historical information about the product, `Reference_List` points to other sources of information about the product that a user may wish to pursue. `Observation_Area` provides information on how the data (or equivalent) were acquired.

[4] Except for documents, in which case each object must be in a separate file.

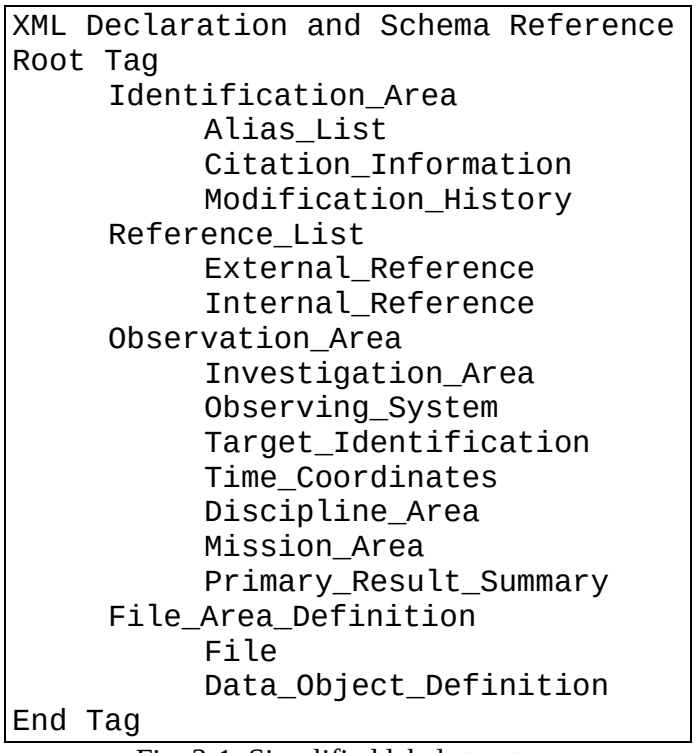


Fig. 3-1. Simplified label structure

Data providers should consult the Information Model for details and the *Data Provider's Handbook* for suggestions on how to construct actual labels. Most of the top-level components have nested classes that are not shown in Figure 3-1.

4 Fundamental Data Structures

There are four fundamental data structures that may be used for archiving data in the PDS. All data products delivered to the PDS must be constructed from one or more of these structures. These four fundamental structures are described using four base classes: Array (used for homogeneous N-dimensional arrays of scalars), Table_Base (used for repeating records of heterogeneous scalars), Parsable_Byte_Stream (a stream of bytes that can be parsed using standardized rules), and Encoded_Byte_Stream (an encoded stream of bytes). All other digital object classes in the PDS are derived from one of these four base classes.

4A Array

The first of the four basic PDS4 structures is the Array. Any data structure that consists of fixed-length rows of homogeneous elements in any number of dimensions must be described using the Array class or one of its subclasses. As two- and three-dimensional (2-D and 3-D, respectively) arrays are among the most commonly used data structures in science, the discussion below will focus on them.

4A.1 Storage Order and Index Order - Definitions

The location of a particular element in a 2-D array is specified using its row number (first index) and column number (second index) when using standard matrix notation. Thus, in the array shown below, the position of the "2" (row 1, column 2) is specified using the notation (1,2), while the position of the "4" (row 2, column 1) is specified as (2,1).

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

An N-dimensional array is always stored in computer memory or in a data file as a linear sequence of numbers. There are two ways to store 2-D array data in linear memory: row-major order and column-major order. In row-major storage, the rows of the array are stored sequentially. In column-major storage, the columns are stored sequentially.

Thus, using the above notation, a row-major 2x3 array would be stored this way in memory:

(1,1), (1,2), (1,3), (2,1), (2,2), (2,3)

or, using the numbers in the example above, the values would be stored as follows:

1 2 3 4 5 6

The same array, in column-major order, would be stored this way:

(1,1), (2,1), (1,2), (2,2), (1,3), (2,3)

or:

1 4 2 5 3 6

PDS uses the terminology *Last_Index_Fastest* as a synonym for 2-D row-major order and *First_Index_Fastest* as a synonym for 2-D column-major order. The PDS terms generalize immediately to N-dimensional arrays whereas the meanings of "row" and "column" become ambiguous. In an N-dimensional array with Last_Index_Fastest, the next to last index varies next to fastest and the second index varies next to slowest.

Last_Index_Fastest (row-major) is the required storage order for PDS4 array data. Note that this does not require modifying data already stored in First_Index_Fastest (column-major) order; only the order in which the axes are defined in the label needs to be switched. See the discussion below under Axis Meaning for more information.

4A.2 Storage and Index Order - Conventions in Popular Software Environments

FORTRAN, IDL, and MATLAB arrays are stored in First_Index_Fastest (column-major) order; all other major programming languages store arrays in Last_Index_Fastest (row-major) order.
Both storage orders were permitted in PDS3 data, but Last_Index_Fastest (row-major) was the default and recommended storage order.

FITS format uses First_Index_Fastest storage order if one assumes that the shape of the array is defined by (NAXIS1, NAXIS2, ...). However, the popular FITS programming libraries cfitsio and pylab describe the arrays as Last_Index_Fastest, *i.e.*, with the shape (..., NAXIS2, NAXIS1).

VICAR indices are listed in First_Index_Fastest order, although this only affects the definitions of keywords N1, N2, N3, and N4. In practice, VICAR mixes together the issues of storage order and axis meaning, using other keywords NS, NL, NB, and ORG.

ISIS exclusively uses First_Index_Fastest storage order.

4A.3 Array Storage Elements

PDS accommodates only binary data in arrays (no character formats). Characteristics of the homogeneous elements (or pixels) in an array must be described using the Element_Array class.

4A.4 Axis Meaning

In an N-dimensional array, each axis needs to be assigned a meaning. Common uses of axes are for spatial coordinates, RGB color, frequency or wavelength, and time. The number of axes in a scientific data file can exceed three, but it rarely does.

PDS uses the terms *Line* and *Sample* to distinguish the two axes of a displayed array. It shares this usage with VICAR and ISIS, but not FITS. The relationship is defined exclusively by the storage order in the file, with the line number increasing more slowly than the sample number.

This means that, for Last_Index_Fastest of a 2-D array, the first index would be the slower varying or the Line dimension, while the second index would be the faster varying or Sample dimension; the

array would have index order (Line, Sample).

In an array with two spatial dimensions and one spectral dimension, PDS uses the term *Band* to designate the spectral axis. A spectral axis may vary faster than both spatial axes, slower than both, or in between the two. An array in which the band axis varies slower than the two spatial axes is typically referred to as *band sequential*. An array in which the band axis varies faster than the two spatial axes is typically identified as *sample interleaved* or *band interleaved by pixel*. An array in which the band axis varies slower than the Sample axis, but faster than the Line axis, is referred to as *line interleaved* or *band interleaved by line*. For Last_Index_Fastest storage, a 3-D band sequential array (sometimes known as a 'cube') would have index order (Band, Line, Sample), a sample interleaved array would have index order (Line, Sample, Band), and a line interleaved array would have index order (Line, Band, Sample).

To maximize the efficiency of playing potentially large movies (Array_3D_Movie), the temporal axis should be the slowest varying axis.

The characteristics of each array axis may be modeled using the Axis_Array class. There must be one Axis_Array class present in the label for each dimension of an array defined by Array_2D or Array_3D or any of their subclasses. Axis_Array is optional (but strongly recommended) for arrays with more dimensions.

4A.5 Display Orientation

Properly defining the orientation in which an image should be presented on a display device is important when an array has two spatial dimensions. Each spatial axis of the array is associated with a direction on the screen. In most image display standards (including this one) the Sample, or faster varying axis, is associated with the horizontal direction on the screen. The Line, or slower varying axis, is associated with the vertical direction.

By almost universal convention, samples are displayed on devices from left to right. There is not, however, unanimity in how the line dimension should be handled. FITS, for example, assumes lines increase from bottom to top; VICAR and ISIS count lines from top to bottom. PDS leaves the choice of line display direction up to the data provider. This information (*e.g.*, down, up) is encoded in an Array_2D_Image label using the Display_2D_Image class (Display_2D_Image is not used for other classes).

4B Table Base

The second of the four basic PDS4 structures is Table_Base. Data structures that consist of fixed-length rows of heterogeneous elements must be described using the Table_Base class or one of its subclasses.[5] At the current time, only two-dimensional tables are permitted.

Conceptually, tabular data consist of named columns containing data values at fixed locations. The data may consist of numbers and character strings including dates, times, and Boolean values; but any one column — also known as a field — contains only values of a single type.

[5] See Section 4C.1 for information on table-like structures with variable length records.

Physically, the data are stored as a sequence of identically structured records where each record may be terminated by a record delimiter (required for Table_Character). Fields within each record are of fixed length and begin at fixed locations. Since both field lengths and record lengths are fixed, field values can be identified by position alone. However, field delimiters may optionally be included.

The data may be represented in binary, ASCII, or UTF-8. Binary tables must be described using the Table_Binary subclass of Table_Base, and ASCII and UTF-8 tables must be described using the Table_Character subclass. These two structures are similar, except that the former contains records described using a Record_Binary class and the latter contains records described using a Record_Character class. Because the overall structure and associations are the same, only Table_Character is discussed in detail. An example Table_Character is below:

```
A-2  S After Deployment   2.3 0.9 7.9 1.2  7.4 0.7<CR><LF>
A-3  R Barnacle Bill      3.2 1.3 3.0 0.5 10.8 1.1<CR><LF>
A-4  S Next to Yogi       3.8 1.5 8.3 1.2  9.1 0.9<CR><LF>
A-5  S Dark Next to Yogi  2.8 1.1 7.5 1.1  8.7 0.9<CR><LF>
A-7  R Yogi               1.7 0.7 5.9 0.9  9.1 0.9<CR><LF>
```

The Record_Character class is used to describe the structure of each record in the table.

4B.1 Fields

There must be one explicitly defined Field_Character class for every field in a Record_Character except when Group_Field_Character can be used. Field definitions within a label must be in the same order as the physical appearance of the fields in the record.

4B.1.1 Field Length

Attribute <field_length> gives the number of bytes in a fixed-width field. Field delimiters (if any) and bracketing double quotes around character strings (if any) are not included in the count.

4B.1.2 Field Formats

Attribute <field_format> provides width and precision information that can be used in validating individual values in character table data. In binary tables, <field_format> can be used as an output format for converting binary numeric values to a character form without losing or overstating significant digits.

The formation rule for a <field_format> value is

```
%[+|-]width[.precision]specifier
```

where square brackets indicate an optional component, “%”is the percent sign (which must precede every field format value), and:

`[+|-]` denotes either a "+" or "-", but never both. The "-" may be used for string fields, to indicate that the string is (or should be) left-justified in the field. This is actually the preferred way to present most string values in character tables, so the

<field_format> value for fields with a data type of ASCII_String will nearly always begin with a "-". Similarly, the "-" denotes left justification for any of the date/time type fields. In PDS4 labels, the "-" prefix is *forbidden* for all numeric fields (integers, floating point numbers, and numbers using scientific notation). The "+" may be used with numeric fields to indicate that an explicit sign is included in the field for input, and should be displayed on output. In PDS4 labels, the "+" is *forbidden* for string fields.

`width` is the potential total width of the field — *i.e.*, the width of the widest value occurring in the field. `width` is an integer indicating the maximum number of characters needed for the complete representation of the largest (in terms of display bytes, not necessarily magnitude) value occuring, or potentially occuring, in the field. This should include bytes for signs, decimal points, and exponents. In the case of string values, it is the maximum width from the first non-blank character to the last non-blank character. It does not include bytes for field delimiters or double quotes (“) around character strings, which are not considered part of the field. In character tables, it *must* be the same as <field_length> for scalar fields.

`width` is separated from `precision` by a decimal point ("."). If there is no `precision` specified, the decimal point must be omitted.

`precision` is the number of digits following the decimal point for real numbers (but is otherwise ignored). `precision` is used in three different ways:

1. For real numbers, it indicates the number of digits to the right of the decimal point.
2. For integers, it indicates that the integer will be zero-padded on the left out to the full field width. For example, the value "`2`" in "`%3.3d`" format is "`002`".
3. For strings, it signifies the maximum number of characters from the actual string value that should be printed. (It is possible in programming, for example, to print no more than the first 10 characters from a string, but require that the output field be left-justified and padded with at least 5 blanks by using a specifier of "`%15.10s`".) In PDS4 labels, if `precision` is included for a string format, it *must* be equal to `width`.

`specifier` is exactly one of the characters in the set `[doxfes]` where

- `d` indicates a decimal integer
- `o` indicates an unsigned octal integer
- `x` indicates an unsigned hexidecimal number
- `f` indicates a floating point number in the format *[-]ddd.ddd*, where the actual number of digits before and after the decimal point is determined by the preceding `width` and `precision` values (note that `width` includes the decimal point and any sign).
- `e` indicates a floating point number in the format *[-]d.ddde+/-dd* where "+/-" stands for exactly one character (either "+" or "-"), there is always exactly one digit to the left of the decimal point, and the number of digits to the

right of the decimal point is determined by the preceding `precision` value (note that the `width` includes all digits, signs, and the decimal point).

`s` indicates a string value. Note that strings should generally be left-justified in fixed width character tables and on output from a binary table, so most <field_format> values ending in "**s**" should begin with "**-**"

4B.2 Groups

Tables are constructed from records, and records are constructed from fields. Repeating sets of fields may be 'grouped' within a record, simplifying their definition.

The Group class defines the set of (repeating) fields and, possibly, (sub) groups. Required group attributes include:

<fields>	the number of scalar fields in the repeating structure
<groups>	the number of (sub)groups in the repeating structure
<repetitions>	the number of repetitions of the repeating structure

Fields and groups may be numbered at each level of nesting using the optional attributes

<field_number>	the position of a field within a series of fields, counting from 1
<group_number>	the position of a group within a series of groups, counting from 1

If two fields within a record are physically separated by one or more groups, they have consecutive field numbers; the fields within the intervening group(s) are numbered separately. Fields within a group separated by one or more (sub)groups, will also have consecutive field numbers. Similarly, group numbering is continuous across intervening fields.

4C Parsable Byte Stream

A parsable byte stream is a stream of bytes, either binary or character, that can be interpreted according to a standard set of rules. For example, a simple ASCII text file can be a parsable byte stream. It consists of a stream of character data; lines are delimited by a standard set of characters (usually the carriage-return line-feed pair or a line-feed by itself). Many different applications are able to parse this format. An HTML file is also a parsable byte stream:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
          "http://www.w3.org/TR/html4/loose.dtd">
<html>
   <body>
   <h1>My First Heading</h1>
   <p>My first paragraph.</p>
   </body>
</html>
```

The above file could be parsed and displayed by any browser programmed to understand the HTML

4.01 standard.

XML labels, comma separated value (CSV) tables, and SPICE kernels are other examples of parsable byte streams. Several formats of data headers are also recognized by PDS as parsable byte streams, including FITS, ISIS, ODL, TIFF, and VICAR headers as well as the PDS Header. Note that 'parsable' is not synonymous with 'human readable.'

The Parsable_Byte_Stream class and its subclasses are used to describe this form of data. The attribute <parsing_standard_id> is used to identify the parsing standard to be used.

4C.1 Delimiter Separated Value Format Description

The delimiter-separated value (DSV, or "spreadsheet") format has been used for some time in a variety of forms for storage and exchange of data between programs and systems. Special cases include the tab-separated value (TSV) and comma-separated value (CSV) formats. This section describes a general DSV format; it is based on RFC 4180 (http://tools.ietf.org/html/rfc4180).

The DSV format is defined as follows:

1. The data comprise one or more records.

2. Each record, including the last, is followed by a record delimiter, the ASCII carriage-return line-feed pair (denoted <CR><LF>), as required for the multipurpose transport of files (RFC 2046; http://www.rfc-editor.org/info/rfc2046). For example:

   ```
   aaa,bbb,ccc<CR><LF>
   zzz,yyy,xxx<CR><LF>
   ```

3. Within each record, there will be one or more fields; fields are separated by field delimiters, and every record has the same number of fields. All field delimiters are the same. There is no field delimiter after the last field and before the record delimiter. The field delimiter must be one of the following characters: comma (,), semi-colon (;), vertical bar (|), or horizontal tab (<HT>). The first occurrence of one of these characters sets the delimiter to be used for all other fields. For example:

   ```
   aaa,bbb,ccc<CR><LF>
   ```

   defines comma (,) as the delimiter; while:

   ```
   aaa|bbb|ccc<CR><LF>
   aaa|b,b|ccc<CR><LF>
   ```

   defines the vertical bar (|) to be the delimiter. Each record above contains three fields. The comma in the second record is part of the second field value.

4. A field may be empty. The interpretation of an empty field will be application and data type dependent. For example:

```
aaa,bbb,ccc<CR><LF>
aaa,,ccc<CR><LF>
```

has an empty field as the second field in the second record.

5. Leading and trailing spaces in a field are considered part of the field. For example:

```
aaa,bbb,   ccc<CR><LF>
zzz,yyy,xxxxxx<CR><LF>
```

contains three spaces in the third field of the first record, making its field length equal to 6. A field is the content between two delimiters, and a value is the information contained within a field. In most cases the field and value are synonymous; in this example, the application determines whether the value has 3 or 6 characters.

6. Fields may optionally be bracketed by a pair of bounding double quotes. The double quotes must be the first and last characters between delimiters; they are not counted in the field length and are not part of the field value. Any characters that are enclosed in double quotes are considered literal (including delimiters, and leading or trailing spaces); double quotes override the default specification of the field delimiter in 3 (above). If bracketing double quotes are used, there may be no double quotes within the field itself. In the example:

```
"aaa,bbb",ccc<CR><LF>
```

the first comma is not treated as a delimiter; it is counted in the field length of the first field and is part of the corresponding value. The second comma in the record defines the field delimiter. This record consists of two fields (with values of aaa,bbb and ccc). In another example:

```
aaa,"   bbb",   ccc<CR><LF>
```

the second and third fields consist of 6 characters each. The spaces in those two fields may be interpreted differently when the application extracts the values. The spaces in the second field are considered to be part of the value.

Double quote usage may vary from record to record. There is no requirement that a particular field be quoted in every record.

A pair of double quotes (,"",) is interpreted as an empty field (length 0).

4C.2 Delimited Tables

The previous section describes the format of delimited tables. This section explains how the characteristics of those tables must be described in labels.

The Table_Delimited class inherits several attributes from the Parsable_Byte_Stream class, and it adds several more.

Each Table_Delimited class requires one Record_Delimited class, which describes the structure of all

records in the delimited table.

Although the individual fields may vary in size from one record to the next, the number of fields, their names, and their data types must remain the same from line to line. There must be one Field_Delimited class present in the label to describe each field in the table record, except when Group_Field_Delimited can be used. Field definitions within the label must be in the same order as the physical appearance of the fields in the record.

The attribute <field_delimiter> must be defined in the Table_Delimited class (and it must be consistent with the provision in Section 4C.1 item 3). Attribute <maximum_field_length> gives the maximum number of bytes in a field. Field delimiters (if any) and bracketing double quotes around character strings (if any) are not included in the count. Values for attribute <field_format> are set as described in Section 4B.1.2.

Repeating sets of fields may be 'grouped' within a record, simplifying their definition, using the Group_Field_Delimited class as described in Section 4B.2.

4D Encoded Byte Stream

The encoded byte stream structure is a byte stream that may only be interpreted after it has been 'decoded' according to some well-known algorithm. For example, 'encoded' data may have been compressed and need to be uncompressed before interpretation. It is PDS policy that only publicly available, open source, widely accepted standards be used for the encoding of data within the PDS; the attribute <encoding_standard_id> identifies the standard.

There are three subclasses of the Encoded Byte Stream class: Encoded_Binary (for files such as PDF and Microsoft Word documents), Encoded_Image (for browse, thumbnail, and document images stored in formats such as GIF, JPEG, and non-raster TIFF), and Encoded_Header (for binary headers such as TIFF headers on raster-formatted TIFF images.

5 Data Types

5A Attribute Value Types

Attribute value types are used to classify the data types of attribute values used in XML labels. These data types are typically specified in class and attribute descriptions in data dictionaries.

Since PDS XML labels contain ASCII (and occasionally UTF-8) characters, the data types in this section do not describe the binary format in which values may be stored by an application, but rather what validation criteria should be applied to the character values as they appear in labels. For example, an ASCII_Integer value should contain only the characters 0-9, "+", and "-"; and an ASCII_String may contain blanks, carriage-returns, tabs, and other white space characters, which may or may not be removed, depending on the data type, before the value is interpreted by a subsequent application.

5A.1 Boolean Types

The PDS boolean data type is based on the primitive boolean type as defined in section 3.2.2 of the World Wide Web Consortium (W3C) *XML Schema Part 2: Datatypes* (W3C, 2004c).

Table 5C-1. Boolean Types

Data Type	Description	Permitted Values
ASCII_Boolean	True/False indicator	`true` and `false` (lower case only), or `1` (true) and `0` (false)

5A.2 Date and Time Types

PDS date and time formats are based on the extended format of cardinal and ordinal date/time strings as defined in ISO 8601:2004. Specifically, all dates and/or times must be represented using one of the two formats below:

`YYYY-MM-DDThh:mm:ss.fffZ` (calendar format)

or

`YYYY-DOYThh:mm:ss.fffZ` (ordinal format)

where

`YYYY`	is the 4-digit year
`MM`	is the 2-digit month (possible values 01-12)
`DD`	is the 2-digit day of month (possible values (01-31)
`DOY`	is the 3-digit day of year (possible values 01-365, or 366 in a leap year)
`hh`	is the 2-digit hour (possible values 00-23)
`mm`	is the 2-digit minute (possible values (00-59)
`ss`	is the 2-digit second (possible values 00-59, or 60 if needed for a leap second)
`fff`	is decimal seconds (number of digits commensurate with precision)

`Z` denotes UTC (may be required or optional depending on type)

- All components (except `fff` and `Z`) must be left padded with zeroes to reach the required number of digits.
- All times must be in the 24-hour clock format
- The ASCII period "." is the only delimiter allowed between the seconds and fractional seconds components.
- The ASCII colon is the only delimiter allowed between the other time components, the ASCII hypen "-" is the only delimiter allowed between date components, and an upper case "T" is the only delimiter allowed between date and time components.
- Delimiters between components must be present if there are values on both sides; they must be omitted if there is not a value on each side.
- Precision may be reduced by dropping any delimiter and all components to its right.
- A time only format may be obtained by dropping "T" and all components to its left.
- Any value not specifically indicated as being UTC by including the "Z" indicator is assumed to be a local time.

Any data preparer who anticipates archiving data with dates earlier than 15 October 1582 or later than 31 December 9999 must consult PDS for special instructions.

Native times, such as from spacecraft clock counters, have formats defined in the appropriate mission or discipline dictionary.

All values for the attributes <start_date_time> and <stop_date_time> must be given in UTC.

The optional variable length decimal fraction of seconds is denoted by `[.fff]`. `[Z]` indicates that the suffix is optional, but PDS strongly encourages its use when the time is given in UTC.

Table 5C-2. Date and Time Types

Data Type	Description	Permitted Values

ASCII_Date	An ASCII date string in either ordinal (DOY) or calendar (YMD) format.	Date value in any of the following forms:
		`YYYY[Z]`
		`YYYY-DOY[Z]`
		`YYYY-MM[Z]`
		`YYYY-MM-DD[Z]`

ASCII_Date_DOY	An ASCII date string in ordinal format.	Date value in either of the following forms:
		`YYYY[Z]`
		`YYYY-DOY[Z]`

ASCII_Date_YMD	An ASCII date string in calendar format.	Date value in any of the following forms:
		`YYYY[Z]`
		`YYYY-MM[Z]`
		`YYYY-MM-DD[Z]`

Table 5C-2. Date and Time Types		
Data Type	**Description**	**Permitted Values**

ASCII_Date_Time	An ASCII date/time string in either ordinal or calendar format.	Date/time value in any of the following forms:[6]
		`YYYY-DOYThh[Z]`
		`YYYY-DOYThh:mm[Z]`
		`YYYY-DOYThh:mm:ss[.fff][Z]`
		`YYYY-MM-DDThh[Z]`
		`YYYY-MM-DDThh:mm[Z]`
		`YYYY-MM-DDThh:mm:ss[.fff][Z]`

ASCII_Date_Time_DOY	An ASCII date/time string in ordinal format.	Date/time value in any of the following forms: [6]
		`YYYY-DOYThh[Z]`
		`YYYY-DOYThh:mm[Z]`
		`YYYY-DOYThh:mm:ss[.fff][Z]`

ASCII_Date_Time_UTC	An ASCII date/time string that must be in Coordinated Universal Time (UTC).	Date/time value in either of the forms: [6]
		`YYYY-MM-DDThh:mm:ss[.fff]Z`
		`YYYY-DOYThh:mm:ss[.fff]Z`

ASCII_Date_Time_YMD	An ASCII date/time string in calendar format.	Date/time value in any of the following forms: [6]
		`YYYY-MM-DDThh[Z]`
		`YYYY-MM-DDThh:mm[Z]`
		`YYYY-MM-DDThh:mm:ss[.fff][Z]`

ASCII_Time	An ASCII time string. May be used for local times on Earth or local solar time on other planets.	Time value in any of the following forms:
		`hh[Z]`
		`hh:mm[Z]`
		`hh:mm:ss[.fff][Z]`

5A.3 Numeric Types

PDS numeric data types are based on a mixture of primitive and derived types defined in (W3C, 2004c) and PDS types derived from these. The specific base type for each data type is indicated in the Description column in the table below.

Table 5C-3. Numeric Types		
Data Type	**Description**	**Permitted Values**

[6] In cases when time information is not available, this format may be truncated to a date string of as little as the year (YYYY).

Table 5C-3. Numeric Types		
Data Type	**Description**	**Permitted Values**
ASCII_Integer	An ASCII character representation of a decimal integer. Based on the derived data type `xs:integer`.	An ASCII string consisting of the digits `0` through `9`, optionally prefixed with a positive "`+`" or negative sign "`-`".
ASCII_NonNegative_Integer	An ASCII character representation of a decimal integer greater than or equal to zero. Based on the derived data type `xs:nonNegativeInteger`.	An ASCII string consisting of the digits `0` through `9`.
ASCII_Real	An ASCII character representation of a real number. Based on the primitive data type `xs:double`.	An ASCII string consisting of a mantissa followed, optionally, by the character `E` or `e`, followed by an exponent. The mantissa must be a decimal number, consisting of a sequence of digits `0` through `9` separated by a period "." as a decimal indicator. The value may be optionally prefixed with a positive "`+`" or negative "`-`" sign. Leading and trailing zeroes are permitted. If the fractional part is zero, the period and following zeroes may be omitted. The exponent must be a signed integer. PDS does not allow positive infinity (INF), negative infinity (-INF) or not-a-number (NaN).
ASCII_Numeric_Base2	An ASCII character representation of a number in binary format. This is a PDS defined data type.	An ASCII string consisting of the characters `0` and `1`. Limited to 255 characters.
ASCII_Numeric_Base8	An ASCII character representation of a number in octal format. This is a PDS defined data type.	An ASCII string consisting of the digits `0` through `7`. Limited to 255 characters.
ASCII_Numeric_Base16	An ASCII character representation of a number in hexadecimal format. Based on the primitive data type `xs:hexBinary`.	An ASCII string consisting of the characters `0` through `9` and `A` through `F` or `a` through `f`. Limited to 255 characters.

Table 5C-3. Numeric Types		
Data Type	**Description**	**Permitted Values**
ASCII_MD5_Checksum	An ASCII representation of a 128-bit hash value calculated using the MD5 algorithm (RFC 1321). This is a PDS defined data type.	An ASCII string consisting of the characters `0` through `9` and `A` through `F` or `a` through `f`. Must be exactly 32 characters in length.

5A.4 String Types

PDS string data types are based on a mixture of primitive and derived types defined in (W3C, 2004c) and PDS types derived from these. The specific base type for each data type is indicated in the Description column in the table below.

Table 5C-4. String Types		
Data Type	**Description**	**Permitted Values**
ASCII_AnyURI	A URI, and its subclasses URN and URL. `xs:anyURI`.	See (W3C, 2004c)
ASCII_DOI	A Digital Object Identifier (DOI) as assigned by members of the International DOI Federation.	See International DOI Federation documentation
ASCII_LID	A PDS logical identifier.	See Section 6D.2
ASCII_LIDVID	A PDS versioned identifier (logical identifier plus version identifier).	See Section 6D.3
ASCII_LIDVID_LID	Either a PDS logical identifier or PDS versioned identifier.	See Sections 6D.2 and 6D.3
ASCII_VID	A PDS version identifier	See Section 6D.3
ASCII_Directory_Path_Name	A directory path in UNIX format.	ASCII string of the form dir1/dir2/ . See Section 6C.2.
ASCII_File_Name	A file name.	ASCII string of the form filename.ext. See Section 6C.1.
ASCII_File_Specification_Name	A directory path and file name (including file name extension) in UNIX format.	ASCII string of the form dir1/dir2/filename.ext. See Section 6C.

Table 5C-4. String Types		
Data Type	**Description**	**Permitted Values**
ASCII_Short_String_Collapsed	An ASCII-encoded text string with white space collapsed — that is, contiguous spaces, line feeds, tabs, and carriage returns have been collapsed into a single ASCII space character and leading and trailing spaces have been removed.	An ASCII string containing no more than 255 characters and including no leading or trailing <SP>, no more than one contiguous <SP>, no <LF>, no <HT>, and no <CR>.
ASCII_Short_String_Preserved	An ASCII-encoded text string with all characters preserved.	An ASCII string of no more than 255 characters.
ASCII_Text_Collapsed	An ASCII-encoded text string of unlimited length with white space collapsed — that is, contiguous spaces, line feeds, tabs, and carriage returns have been collapsed into a single ASCII space character and leading and trailing spaces have been removed.	An ASCII string containing no leading or trailing <SP>, no more than one contiguous <SP>, no <LF>, no <HT>, and no <CR>.
ASCII_Text_Preserved	An ASCII-encoded text string of unlimited length with all characters preserved.	An ASCII string.
UTF8_Short_String_Collapsed	A UTF-8 encoded text string with white space collapsed (see ASCII_Short_String_Collapsed above).	A UTF-8 string containing no more than 255 bytes and including no leading or trailing <SP>, no more than one contiguous <SP>, no <LF>, no <HT>, and no <CR>.
UTF8_Short_String_Preserved	A UTF-8 encoded text string with all characters preserved.	A UTF-8 string of no more than 255 bytes.
UTF8_Text_Collapsed	A UTF-8 encoded text string with white space collapsed (see ASCII_Short_String_Collapsed above).	A UTF-8 string including no leading or trailing <SP>, no more than one contiguous <SP>, no <LF>, no <HT>, and no <CR>.
UTF8_Text_Preserved	A UTF-8 encoded text string with all characters preserved.	A UTF-8.

5B Character Data Types

Character data types are used to describe the data formats of character fields in tables.

The values described in section 5A for Boolean Types (Section 5A.1), Date and Time Types (Section 5A.2), and Numeric Types (Section 5A.3) may also be used for field descriptions in tables. Table character fields should use the data types described in the table below.

Table 5B-1. Character Data Types

Data Types	Description	Permitted Values
ASCII_AnyURI	A URI, and its subclasses URN and URL. `xs:anyURI`	See (W3C, 2004c)
ASCII_DOI	A Digital Object Identifier (DOI) as assigned by members of the International DOI Federation.	See International DOI Federation documentation
ASCII_File_Name	A file name.	ASCII string of the form: filename.ext. See Section 6C.1.
ASCII_File_Specification_Name	A directory path and file name (including file name extension) in UNIX format.	ASCII string of the form: dir1/dir2/filename.ext. See Section 6C.
ASCII_LID	A PDS logical identifier.	See Section 6D.2
ASCII_LIDVID	A PDS versioned identifier (logical identifier plus version identifier).	See Section 6D.3
ASCII_VID	A PDS version identifier	See Section 6D.3
ASCII_String	An ASCII string	An ASCII string.
UTF8_String	A UTF-8 string	A UTF8 string.

5C Binary Data Types

Binary data types are used to describe data formats of fields in binary tables and array elements in arrays.

5C.1 Integers

5C.1.1 Signed LSB Integers

This section describes signed integers stored in Least Significant Byte (LSB) first (also known as 'little-endian') order. In this section the following definitions apply:

b0-b7 Arrangement of bytes as they appear when reading a file (*e.g.*, read byte b0 first, then b1, b2 and b3, up through b7)

i-sign Integer sign bit – bit 7 in the highest order byte

i0-i7 Arrangement of bytes in the integer, from lowest order to highest order. The bits within each byte are interpreted from right to left (*e.g.*, lowest value = bit 0, highest value = bit 7), in the following way:

8-byte integers (Fig. 5C-1):
In i0, bits 0-7 represent 2^0 through 2^7
In i1, bits 0-7 represent 2^8 through 2^{15}
In i2, bits 0-7 represent 2^{16} through 2^{23}
In i3, bits 0-7 represent 2^{24} through 2^{31}
In i4, bits 0-7 represent 2^{32} through 2^{39}
In i5, bits 0-7 represent 2^{40} through 2^{47}
In i6, bits 0-7 represent 2^{48} through 2^{55}
In i7, bits 0-6 represent 2^{56} through 2^{62}

4-byte integers (Fig. 5C-2):
In i0, bits 0-7 represent 2^0 through 2^7
In i1, bits 0-7 represent 2^8 through 2^{15}
In i2, bits 0-7 represent 2^{16} through 2^{23}
In i3, bits 0-6 represent 2^{24} through 2^{30}

2-byte integers (Fig. 5C-3):
In i0, bits 0-7 represent 2^0 through 2^7
In i1, bits 0-6 represent 2^8 through 2^{14}

1-byte integer (Fig. 5C-4):
In i0, bits 0-6 represent 2^0 through 2^6

All negative values are represented in two's complement.

data_type = SignedLSB8

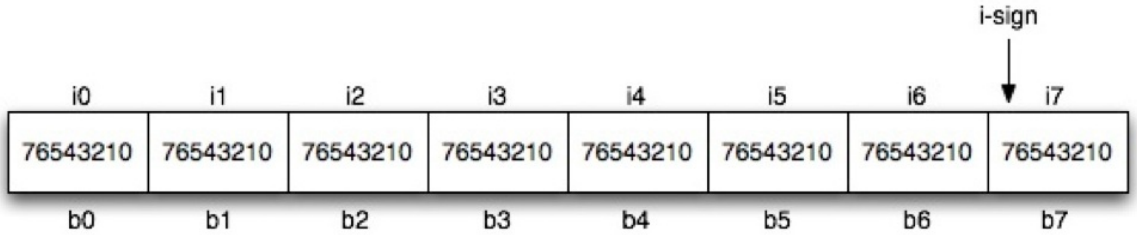


Fig. 5C-1. Signed 8-byte integer, least significant byte first

data_type = SignedLSB4

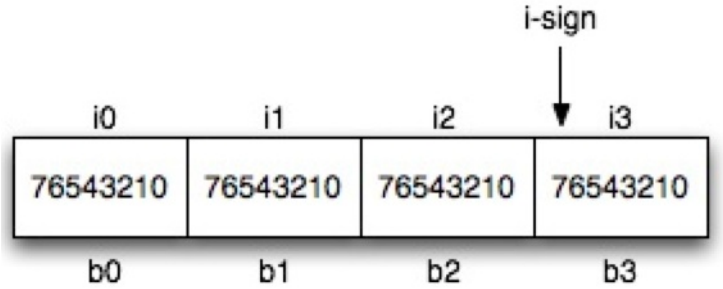


Fig. 5C-2. Signed 4-byte integer, least significant byte first

data_type = SignedLSB2

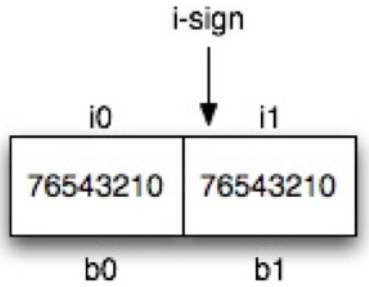


Fig. 5C-3. Signed 2-byte integer, least significant byte first

data_type = SignedByte

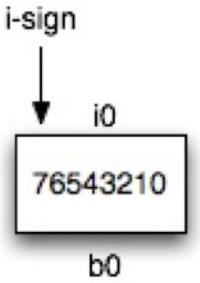


Fig. 5C-4. Signed 1-byte integer

5C.1.2 Unsigned LSB Integers

This section describes unsigned integers stored in LSB format. In this section the following definitions apply:

b0 – b7 Arrangement of bytes as they appear when reading a file (*e.g.*, read byte b0 first, then b1, b2 and b3, up through b7)

i0 – i7 Arrangement of bytes in the integer, from lowest order to highest order. The bits within each byte are interpreted from right to left (*e.g.*, lowest value = bit 0, highest value = bit 7), in the following way:

8-byte integers (Fig. 5C-5):
In i0, bits 0-7 represent 2^0 through 2^7

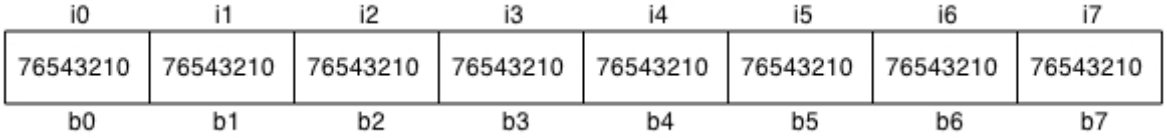
In i1, bits 0-7 represent 2^8 through 2^{15}
In i2, bits 0-7 represent 2^{16} through 2^{23}
In i3, bits 0-7 represent 2^{24} through 2^{31}
In i4, bits 0-7 represent 2^{32} through 2^{39}
In i5, bits 0-7 represent 2^{40} through 2^{47}
In i6, bits 0-7 represent 2^{48} through 2^{55}
In i7, bits 0-6 represent 2^{56} through 2^{63}

4-byte integers (Fig. 5C-6):
In i0, bits 0-7 represent 2^0 through 2^7
In i1, bits 0-7 represent 2^8 through 2^{15}
In i2, bits 0-7 represent 2^{16} through 2^{23}
In i3, bits 0-6 represent 2^{24} through 2^{31}

2-byte integers(Fig. 5C-7):
In i0, bits 0-7 represent 2^0 through 2^7
In i1, bits 0-6 represent 2^8 through 2^{15}

1-byte integers (Fig. 5C-8):
In i0, bits 0-7 represent 2^0 through 2^7

data_type = UnsignedLSB8



Fig, 5C-5. Unsigned 8-byte integer, least significant byte first.

data_type = UnsignedLSB4

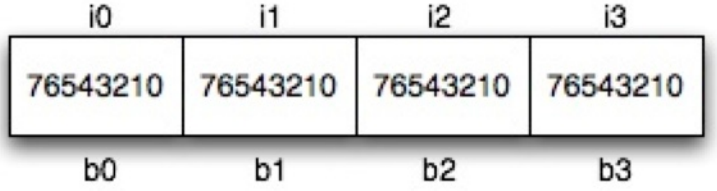


Fig. 5C-6. Unsigned 4-byte integer, least significant byte first

data_type = UnsignedLSB2

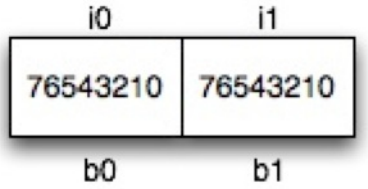


Fig. 5C-7. Unsigned 2-byte integer, less significant byte first

data_type = UnsignedByte

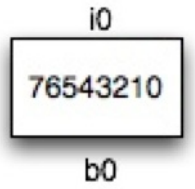


Fig. 5C-8. Unsigned 1-byte integer

5C.1.3 Signed MSB Integers

This section describes the signed integers stored in Most Significant Byte (MSB) first (also known as 'big-endian') order. In this section the following definitions apply:

b0 - b7 Arrangement of bytes as they appear when read from a file (*e.g.*, read b0 first, then b1, b2, and b3, up through b7)

i-sign Integer sign bit – bit 7 in the highest order byte

i0 - i7 Arrangement of bytes in the integer, from lowest order to highest order. The bits within each byte are interpreted from right to left (*e.g.*, lowest value = bit 0, highest value = bit 7) in the following way:

8-byte integers (Fig. 5C-9):
In i0, bits 0-7 represent 2^0 through 2^7
In i1, bits 0-7 represent 2^8 through 2^{15}
In i2, bits 0-7 represent 2^{16} through 2^{23}
In i3, bits 0-7 represent 2^{24} through 2^{31}
In i4, bits 0-7 represent 2^{32} through 2^{39}
In i5, bits 0-7 represent 2^{40} through 2^{47}
In i6, bits 0-7 represent 2^{48} through 2^{55}
In i7, bits 0-6 represent 2^{56} through 2^{62}

4-byte integers (Fig. 5C-10):
In i0, bits 0-7 represent 2^0 through 2^7
In i1, bits 0-7 represent 2^8 through 2^{15}
In i2, bits 0-7 represent 2^{16} through 2^{23}
In i3, bits 0-6 represent 2^{24} through 2^{30}

2-byte integers (Fig. 5C-11):
In i0, bits 0-7 represent 2^0 through 2^7
In i1, bits 0-6 represent 2^8 through 2^{14}

1-byte integers (Fig. 5C-12):
In i0, bits 0-6 represent 20 through 26

All negative values are represented in two's complement.

data_type = SignedMSB8

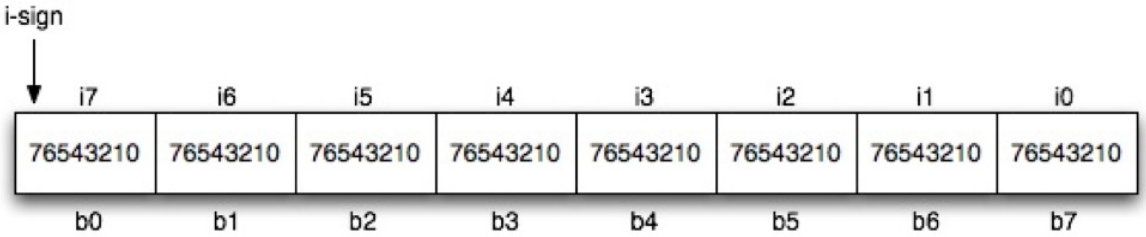


Fig. 5C-9. Signed 8-byte integer, most significant byte first

data_type = SignedMSB4

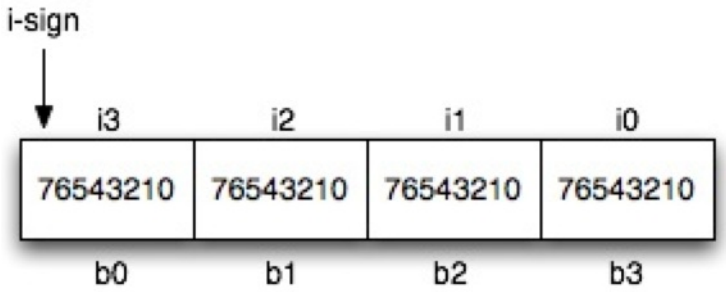


Fig. 5C-10. Signed 4-byte integer, most significant byte first

data_type = SignedMSB2

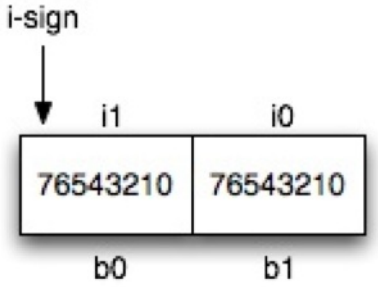


Fig. 5C-11. Signed 2-byte integer, more significant byte first

The signed 1-byte integer is shown in Figure 5C-4.

5C.1.4 Unsigned MSB Integers

This section describes unsigned integers stored in MSB format. In this section the following definitions apply:

b0 – b7 Arrangement of bytes as they appear when read from a file (*e.g.*, read b0 first, then b1, b2, and b3, up through b7)

i0 – i7 Arrangement of bytes in the integer, from lowest order to highest order. The bits within each byte are interpreted from right to left (*e.g.*, lowest value = bit 0, highest value = bit 7) in the following way:

8-byte integers (Fig. 5C-13):

In i0, bits 0-7 represent 2^0 through 2^7
In i1, bits 0-7 represent 2^8 through 2^{15}
In i2, bits 0-7 represent 2^{16} through 2^{23}

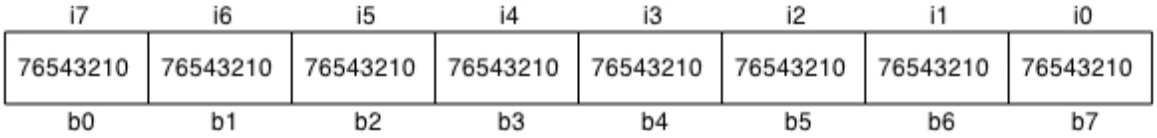
In i3, bits 0-7 represent 2^{24} through 2^{31}
In i4, bits 0-7 represent 2^{32} through 2^{39}
In i5, bits 0-7 represent 2^{40} through 2^{47}
In i6, bits 0-7 represent 2^{48} through 2^{55}
In i7, bits 0-6 represent 2^{56} through 2^{63}

4-byte integers (Fig. 5C-14):
In i0, bits 0-7 represent 2^{0} through 2^{7}
In i1, bits 0-7 represent 2^{8} through 2^{15}
In i2, bits 0-7 represent 2^{16} through 2^{23}
In i3, bits 0-7 represent 2^{24} through 2^{31}

2-byte integers (Fig. 5C-15):
In i0, bits 0-7 represent 2^{0} through 2^{7}
In i1, bits 0-7 represent 2^{8} through 2^{15}

1-byte integers (Fig. 5C-16):
In i0, bits 0-7 represent 2^{0} through 2^{7}

data_type = UnsignedMSB8



Fig, 5C-12. Unsigned 8-byte integer, most significant byte first.

data_type = UnsignedMSB4

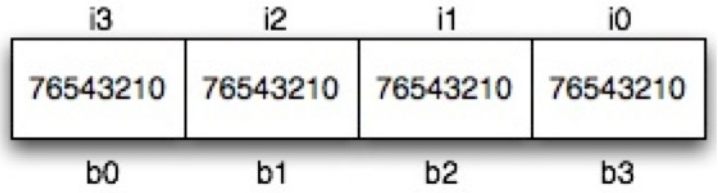


Fig. 5C-13. Unsigned 4-byte integer, most significant byte first

data_type = UnsignedMSB2

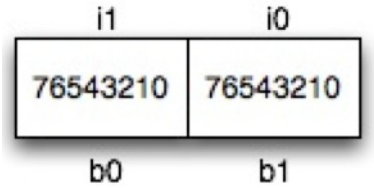


Fig. 5C-14. Unsigned 2-byte integer, more significant byte first

The unsigned 1-byte integer is shown in Figure 5C-8.

5C.2 Reals

This section describes the internal format of IEEE-format floating-point numbers. In this section the following definitions apply:

b0 – b7 Arrangement of bytes as they appear when read from a file (*e.g.*, read b0 first, then b1, b2, b3, etc.)

m-sign Mantissa sign bit

e0 -e1 Arrangement of the portions of the bytes that make up the exponent, from lowest order to highest order. The bits within each byte are interpreted from right to left (*e.g.*, lowest value = rightmost bit in the exponent part of the byte, highest value = leftmost bit in the exponent part of the byte) in the following way:

8-bytes (double precision):
In e0, bits 4-7 represent 2^0 through 2^3
In e1, bits 0-6 represent 2^4 through 2^{10}

Exponent bias = 1023

4-bytes (single precision):
In e0, bit 7 represents 2^0
In e1, bits 0-6 represent 2^1 through 2^7

Exponent bias = 127

m0 – m6 Arrangement of the portions of the bytes that make up the mantissa, from highest order fractions to the lowest order fraction. The order of the bits within each byte progresses from left to right, with each bit representing a fractional power of two, in the following way:

8-bytes (double precision):
In m0, bits 3-0 represent $1/2^1$ through $1/2^4$
In m1, bits 7-0 represent $1/2^5$ through $1/2^{12}$
In m2, bits 7-0 represent $1/2^{13}$ through $1/2^{20}$
In m3, bits 7-0 represent $1/2^{21}$ through $1/2^{28}$
In m4, bits 7-0 represent $1/2^{29}$ through $1/2^{36}$
In m5, bits 7-0 represent $1/2^{37}$ through $1/2^{44}$
In m6, bits 7-0 represent $1/2^{45}$ through $1/2^{52}$

4-bytes (single precision):
In m0, bits 6-0 represent $1/2^1$ through $1/2^7$
In m1, bits 7-0 represent $1/2^8$ through $1/2^{15}$
In m2, bits 7-0 represent $1/2^{16}$ through $1/2^{23}$

The stored value may be recovered as follows:

$$1.mantissa \times 2^{(exponent-bias)}$$

Note that the integer part ("1.") is implicit in all formats as described above. In all cases the exponent is stored as an unsigned, biased integer (that is, the stored exponent value - bias value = true exponent).

data_type = IEEE754LSBDouble

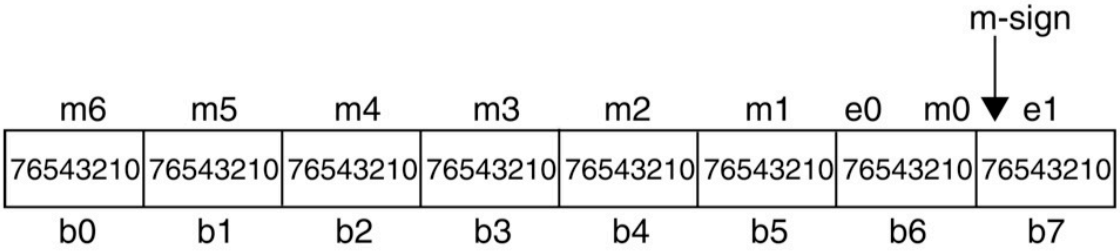


Fig. 5C-15. IEEE 754 double precision real, least significant byte first.

data_type = IEEE754LSBSingle

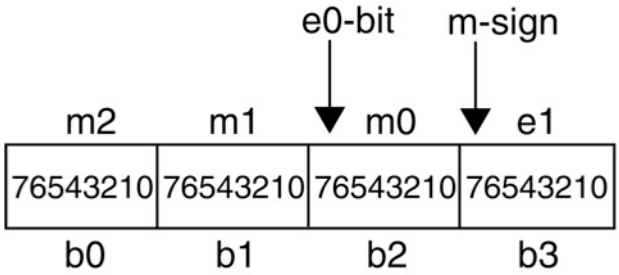


Fig. 5C-16. IEEE 754 single precision real, least significant byte first.

data_type = IEEE754MSBDouble

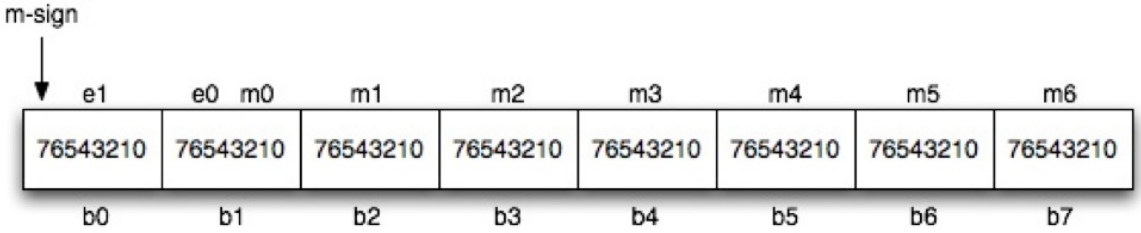


Fig. 5C-17. IEEE 754 double precision real, most significant byte first.

data_type = IEEE754MSBSingle

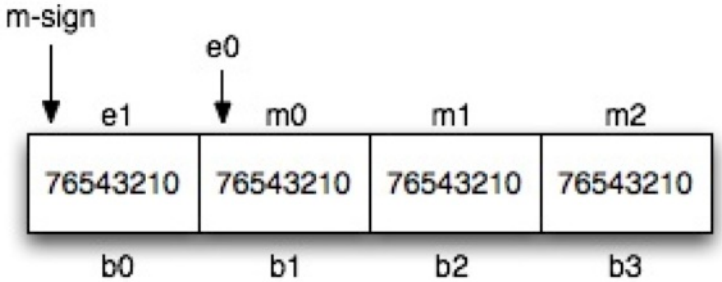


Fig. 5C-18. IEEE 754 single precision real, most significant byte first

5C.3 Complex

IEEE complex numbers consist of two IEEE Real format numbers of the same precision, contiguous in memory. The first number represents the real part and the second the imaginary part of the complex value.

data_type = ComplexLSB16

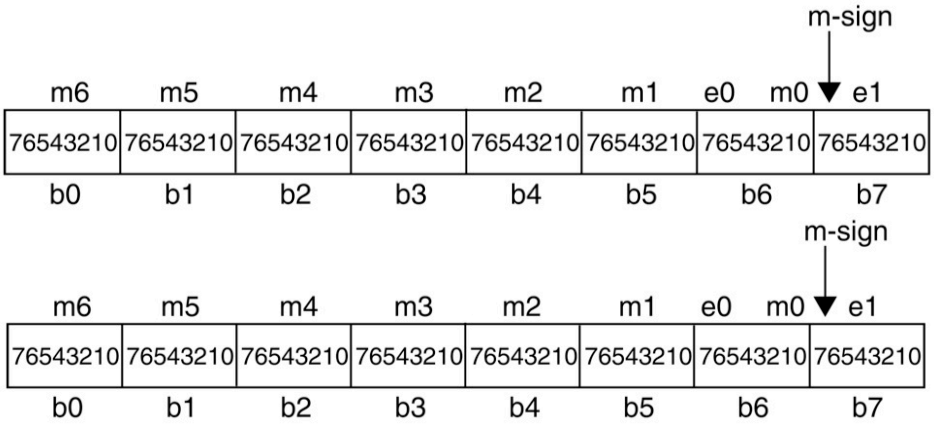


Fig. 5C-19. Double precision complex, least significant bytes first; real component is shown above, followed by imaginary component (below)

data_type = ComplexLSB8

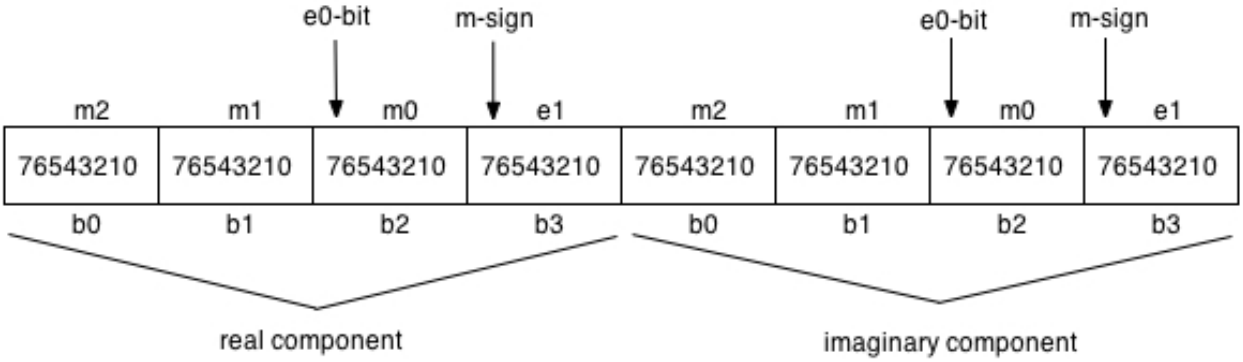


Fig. 5C-20. Single precision complex, least significant bytes first.

data_type = ComplexMSB16

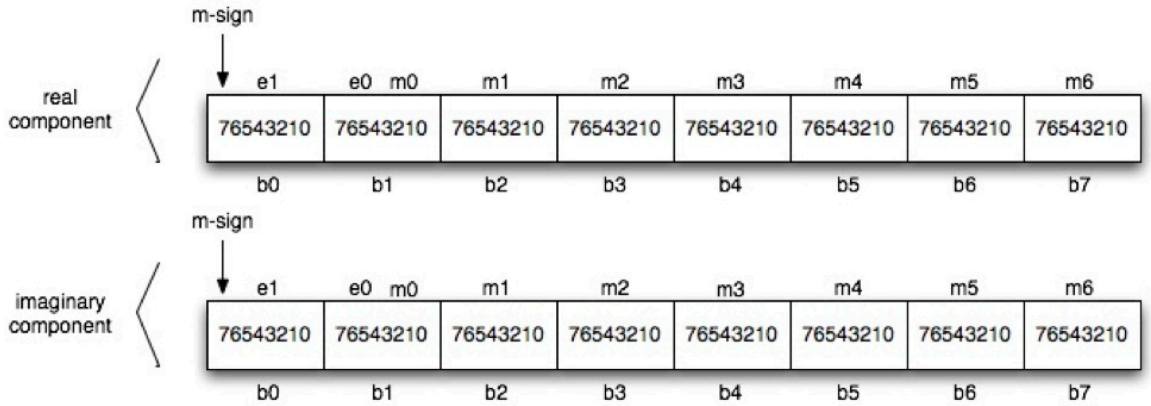


Fig. 5C-21. Double precision complex, most significant bytes first.

data_type = ComplexMSB8

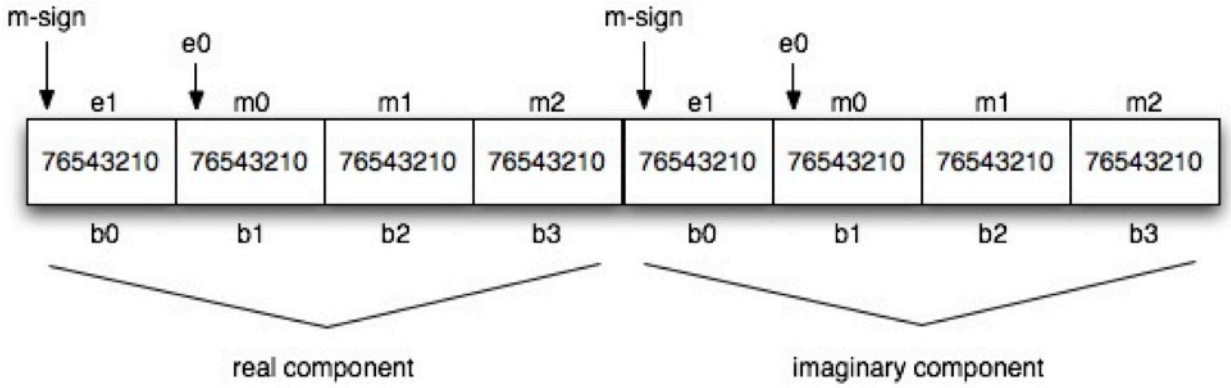


Fig. 5C-22. Single precision complex, most significant bytes first.

5C.4 Bit Strings

This section describes signed and unsigned integers stored as strings of contiguous bits that do not align with byte boundaries. PDS places no constraints on location or length of bit strings; only examples of these structures are shown here.

In the examples below, b0 is the first byte read, and b1 is the second; bits are numbered right to left within each byte.

Fig. 5C-23 shows an example signed bit string that extends from bit 3 of b0 to bit 6 of b1 (six bits total). The sign bit is bit 3 of b0; if the sign bit is "0" the value is positive, if negative, the value is negative and it is represented in two's complement notation.

In b0, bits 2-0 represent 2^4, 2^3, and 2^2, respectively.
In b1, bits 7-6 represent 2^1 and 2^0

Fig. 5C-24 shows an example unsigned bit string extending from bit 2 of b0 to bit 6 of b1 (5 bits total.

data_type = SignedBitString

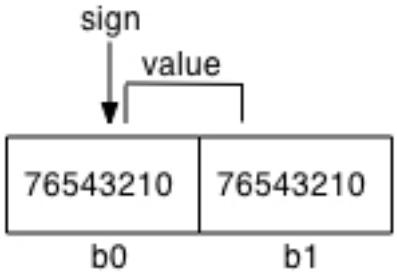


Fig. 5C-23. Signed bit string

data_type = UnsignedBitString

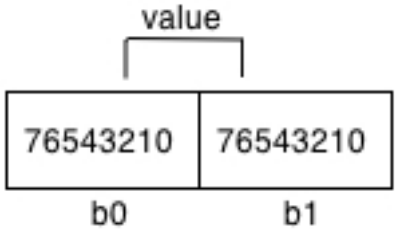


Fig. 5C-24. Unsigned bit string

6A Character Sets

PDS recognizes several character sets.

6A.1 7-Bit ASCII Character Set (also known as IsBasicLatin)

The ASCII character set is defined in octal, hexadecimal, and decimal as follows:

Octal - Character

```
000 NUL  001 SOH  002 STX  003 ETX  004 EOT  005 ENQ  006 ACK  007 BEL
010 BS   011 HT   012 NL   013 VT   014 NP   015 CR   016 SO   117 SI
020 DLE  021 DC1  022 DC2  023 DC3  024 DC4  025 NAK  026 SYN  027 ETB
030 CAN  031 EM   032 SUB  033 ESC  034 FS   035 GS   036 RS   037 US
040 SP   041 !    042 "    043 #    044 $    045 %    046 &    047 '
050 (    051 )    052 *    053 +    054 ,    055 -    056 .    057 /
060 0    061 1    062 2    063 3    064 4    065 5    066 6    067 7
070 8    071 9    072 :    073 ;    074 <    075 =    076 >    077 ?
100 @    101 A    102 B    103 C    104 D    105 E    106 F    107 G
110 H    111 I    112 J    113 K    114 L    115 M    116 N    117 O
120 P    121 Q    122 R    123 S    124 T    125 U    126 V    127 W
130 X    131 Y    132 Z    133 [    134 \    135 ]    136 ^    137 _
140 `    141 a    142 b    143 c    144 d    145 e    146 f    147 g
150 h    151 i    152 j    153 k    154 l    155 m    156 n    157 o
160 p    161 q    162 r    163 s    164 t    165 u    166 v    167 w
170 x    171 y    172 z    173 {    174 |    175 }    176 ~    177 DEL
```

Hexadecimal - Character

```
00 NUL   01 SOH   02 STX   03 ETX   04 EOT   05 ENQ   06 ACK  07 BEL
08 BS    09 HT    0A NL    0B VT    0C NP    0D CR    0E SO   0F SI
10 DLE   11 DC1   12 DC2   13 DC3   14 DC4   15 NAK   16 SYN  17 ETB
18 CAN   19 EM    1A SUB   1B ESC   1C FS    1D GS    1E RS   1F US
20 SP    21 !     22 "     23 #     24 $     25 %     26 &    27 '
28 (     29 )     2A *     2B +     2C ,     2D -     2E .    2F /
30 0     31 1     32 2     33 3     34 4     35 5     36 6    37 7
38 8     39 9     3A :     3B ;     3C <     3D =     3E >    3F ?
40 @     41 A     42 B     43 C     44 D     45 E     46 F    47 G
48 H     49 I     4A J     4B K     4C L     4D M     4E N    4F O
50 P     51 Q     52 R     53 S     54 T     55 U     56 V    57 W
58 X     59 Y     5A Z     5B [     5C \     5D ]     5E ^    5F _
60 `     61 a     62 b     63 c     64 d     65 e     66 f    67 g
68 h     69 i     6A j     6B k     6C l     6D m     6E n    6F o
70 p     71 q     72 r     73 s     74 t     75 u     76 v    77 w
78 x     79 y     7A z     7B {     7C |     7D }     7E ~    7F DEL
```

Decimal - Character

```
  0 NUL    1 SOH    2 STX    3 ETX    4 EOT    5 ENQ    6 ACK    7 BEL
  8 BS     9 HT    10 NL    11 VT    12 NP    13 CR    14 SO    15 SI
 16 DLE   17 DC1   18 DC2   19 DC3   20 DC4   21 NAK   22 SYN   23 ETB
```

```
 24 CAN  25 EM    26 SUB   27 ESC   28 FS    29 GS    30 RS    31 US
 32 SP   33 !     34 "     35 #     36 $     37 %     38 &     39 '
 40 (    41 )     42 *     43 +     44 ,     45 -     46 .     47 /
 48 0    49 1     50 2     51 3     52 4     53 5     54 6     55 7
 56 8    57 9     58 :     59 ;     60 <     61 =     62 >     63 ?
 64 @    65 A     66 B     67 C     68 D     69 E     70 F     71 G
 72 H    73 I     74 J     75 K     76 L     77 M     78 N     79 O
 80 P    81 Q     82 R     83 S     84 T     85 U     86 V     87 W
 88 X    89 Y     90 Z     91 [     92 \     93 ]     94 ^     95 _
 96 `    97 a     98 b     99 c    100 d    101 e    102 f    103 g
104 h   105 i    106 j    107 k    108 l    109 m    110 n    111 o
112 p   113 q    114 r    115 s    116 t    117 u    118 v    119 w
120 x   121 y    122 z    123 {    124 |    125 }    126 ~    127 DEL
```

where the following are non-printing, control characters:

```
NUL = null character
SOH = start of header
STX = start of text
ETX = end of text
EOT = end of transmission
ENQ = enquiry
ACK = acknowledgement
BEL = bell
BS  = backspace
HT  = horizontal tab
LF  = line feed
VT  = vertical tab
FF  = form feed
CR  = carriage return
SO  = shift out
SI  = shift in
DLE = data link escape
DC1 = device 1 control
DC2 = device 2 control
DC3 = device 3 control
DC4 = device 4 control
NAK = negative acknowledgement
SYN = synchronous idle
ETB = end transmission block
CAN = cancel
EM  = end of medium
SUB = substitute
ESC = escape
FS  = file separator
GS  = group separator
RS  = record separator
US  = unit separator
DEL = delete
```

6A.2 ASCII Alphanumeric Character Set

The ASCII alphanumeric character set includes the upper and lower case ASCII letters and the ASCII digits.

upper case letters	A-Z	ASCII 0x41 to 0x5a

lower case letters	a-z	ASCII 0x61 to 0x7a
digits	0-9	ASCII 0x30 to 0x39

6A.3 ASCII Printable Character Set

The ASCII printable character set is the set in Section 6A.1 less the non-printing, control characters. The hexadecimal equivalents are 0x20 to 0x7E, inclusive.

6B Namespace

A *namespace* is a context within which attributes and classes may be defined; it is managed by a *steward*. A namespace may have only one steward, but a steward is allowed to manage more than one namespace. PDS is a *registration authority*; it oversees several namespaces, each closely controlled by a steward; the PDS Engineering Node (EN), acting on behalf of the PDS registration authority, coordinates the stewards.

There is a 'common' namespace managed by EN (under the steward name 'pds'), and each PDS discipline node manages its own namespace. Typically, each planetary mission has a namespace; additionally, some areas of study ('disciplines', such as cartography) have namespaces.

Table 6B.1 shows example namespaces recognized by PDS, along with their Uniform Resource Identifiers (URIs) and their stewards

6B.1 Namespace Creation and Use

To ensure that namespaces are unique across PDS, EN creates them in consultation with the discipline nodes. Typically namespaces are created for new investigations (such as missions) or science specialties. When circumstances appear to warrant the creation of a new namespace, a data provider should confer with the consulting discipline node. Discipline nodes should also be contacted when there is a question about which namespace to use or how to reference a namespace.

6B.2 Formation of Namespace IDs:

The following rules govern creation of namespace identifiers; namespace_id values must be:

- unique across the PDS.
- no longer than 16 characters.
- constructed from ASCII lower case letters, digits, and dash:

a-z	ASCII 0x61 through 0x7A
0-9	ASCII 0x30 to 0x39
-	ASCII 0x7E

,

6B.3 Formation of Namespace URIs:

The following rules govern creation of namespace Universal Resource Identifiers (URIs) in PDS; they must:

- meet `xs:anyURI` requirements
- be unique across the PDS
- be lower case
- begin with "http://" irrespective of whether the namespace is within the PDS domain
- begin with "http://pds.nasa.gov/pds4/" if the namespace is within the PDS domain

Table 6B.1: Example Namespaces Recognized by PDS

Namespace ID	Uniform Resource Identifier	Steward
Global Namespaces		
xs	http://www.w3.org/2001/XMLSchema XML	World Wide Web Consortium (W3C)
xsi	http://www.w3.org/2001/XMLSchema-instance	World Wide Web Consortium (W3C)
sch	http://purl.oclc.org/dsdl/schematron	Schematron.com
PDS (Common) Namespace		
pds	http://pds.nasa.gov/pds4/pds	PDS Engineering Node
PDS Discipline and Discipline Node Namespaces		
atm	http://pds.nasa.gov/pds4/atm	PDS Atmospheres Node
geo	http://pds.nasa.gov/pds4/geo	PDS Geosciences Node
img	http://pds.nasa.gov/pds4/img	PDS Imaging Node
naif	http://pds.nasa.gov/pds4/naif	Navigation and Ancillary Information Facility
ppi	http://pds.nasa.gov/pds4/ppi	PDS Planetary Plasma Interactions Node
rings	http://pds.nasa.gov/pds4/rings	PDS Planetary Rings Node
rs	http://pds.nasa.gov/pds4/rs	PDS Radio Science Advisor
sbn	http://pds.nasa.gov/pds4/sbn	PDS Small Bodies Node
cart	http://pds.nasa.gov/pds4/cart	PDS Imaging Node
PDS Mission Namespaces		
mpf	http://pds.nasa.gov/pds4/mpf	Mars Pathfinder
ldex	http://pds.nasa.gov/pds4/ldex	Lunar Atmosphere and Dust Environment Explorer (LADEE)

6C File and Directory Naming

Although modern operating systems are permissive when it comes to file and directory names, there are good reasons for assigning names conservatively. For example, characters that are legal on one system may need to be 'escaped' on another. Similarly, 255 character names, while permitted, are often inconvenient for users who cannot distinguish among similarly constructed, lengthy names in displays and lists. Also, the concatenation of long file and/or directory names may exceed limits for a path name once all components have been combined. Finally, although much data management is carried on by machines, humans will find intuitive names to be much easier to grasp should they need to inspect an archive manually.

In the following sections "prohibited" means "not allowed" while "reserved" means that the name must be employed in a way that is consistent with PDS usage. For example, "geometry" may only be used in a base name or directory name if the content of the file or directory has an obvious connection to geometry. Case variations of prohibited and reserved names below are similarly constrained.

The asterisk "*" is a wild card denoting zero or more additional characters, and square brackets enclose options.

6C.1 File Names

6C.1.1 Rules

The following rules govern selection of file names. File names must:

- be unique within directories.
- be no longer than 255 characters (and be commensurate with other length limits, such as for path names)
- be case-insensitive; for example, "MyFile.txt" and "myfile.txt" are not permitted in the same directory.
- be constructed from the character set

A-Z	ASCII 0x41 through 0x5A,
a-z	ASCII 0x61 through 0x7A,
0-9	ASCII 0x30 through 0x39,
dash "-"	ASCII 0x2D,
underscore "_"	ASCII 0x5F, and
period "."	ASCII 0x2E.

- not begin or end with a dash, underscore, or period.

6C.1.2 File Name Extensions

File names must include at least one period followed by an *extension*. File names may have more than one period, but PDS will consider all periods other than the final one to be part of the *base name*.

6C.1.3 Prohibited File Names

The following file names have specific purposes on some operating systems; their use within PDS archives is prohibited.

```
a.out                core
```

6C.1.4 Reserved File Names

The following file names are reserved for specific purposes within PDS archives and may not be used otherwise:

```
bundle*.xml                          readme*.[html,txt]
catalog*.xml                         pds4_pds*.[sch,xsd,xml]
collection*.[xml,tab]
```

For example, file names starting with "`readme`" and ending with either "`.html`" or "`.txt`" are reserved while "`readme.abc`" is not.

6C.1.5 Prohibited Base Names

Because the following base names have specific purposes on some operating systems, their use within PDS archives is prohibited.

```
aux          com5         con          lpt5         nul
com1         com6         lpt1         lpt6         prn
com2         com7         lpt2         lpt7
com3         com8         lpt3         lpt8
com4         com9         lpt4         lpt9
```

6C.1.6 Reserved Base Name Components

The following terms and abbreviations are reserved for use as components of base names:

```
browse                 geometry               node
calibration            instrument_host        resource
collection             instrument             schema
context                investigation          spice_kernels
data                   inventory              target
document               miscellaneous
```

6C.1.7 Reserved File Name Extensions

The file name extensions listed in Table 6C-1 are reserved. A brief description for each is provided in the table. The presence of any extension in this table does not imply that the

associated format is acceptable in every (or any) collection.

Table 6C-1 – Reserved File Name Extensions	
Extension	**Description**
bc	SPICE binary CK file (spacecraft and instrument orientation data)
bdb	SPICE binary EK file, data base component (event data base data)
bds	SPICE binary DSK file (natural body digital shape data)
bep	SPICE binary EK file, plan (science plan data)
bes	SPICE binary EK file, sequence component (event sequence data)
bpc	SPICE binary PCK file (high-accuracy natural body rotation data)
bsp	SPICE binary SPK file (trajectory and ephemeris data)
csv	delimited tabular data file (comma separated value)
dat	generic binary data file (used for files containing different types of digital data)
doc,docx	Microsoft Word file
fit,fts	Flexible Image Transport System file
gif	Graphics Interchange Format file
hdr	generic header file, usually character data
htm,html	HyperText Markup Language formatted file
img	raster formatted image data (may contain headers)
jp2	JPEG 2000 formatted file
jpg,jpeg	Joint Photographic Experts Group formatted file
lbl	PDS3 or other format label
nrb	SPICE text orbit number file, ascending or descending node numbering (orbit start and stop times)
orb	SPICE text orbit number file, periapsis or apoapsis numbering (orbit start and stop times)
pdf	Portable Document Format
sch	Schematron file
tab	fixed-width tabular data file
ten	SPICE text EK file, notebook component (observation notes data)
tep	SPICE text EK file, plan component (science plan data)
tf	SPICE text FK file (reference frame definitions)
ti	SPICE text IK file (instrument parameters and FOV definitions)
tif,tiff	Tagged Image Format File
tls	SPICE text LSK file (leapsecond information)
tm	SPICE text MK file (meta-kernels listing kernels to be used together)
tpc	SPICE text PCK file (natural body rotation and size/shape constants)
tsc	SPICE text SCLK file (spacecraft clock correlation data)
txt	plain text file
vic	Video Image Communication and Retrieval file
xml	Extensible Markup Language formatted file
xsd	XML Schema file
zip	Zip compressed archive file

6C.2 Directories

6C.2.1 Rules

The following rules govern selection of directory names. Directory names must:

- be unique within parent directories.
- be no longer than 255 characters (and be commensurate with other length limits, such as for path names).
- be case-insensitive; for example, “MyDirectory” and “mydirectory” are not permitted in the same parent directory.
- be constructed from the character set

A-Z	ASCII 0x41 through 0x5A,
a-z	ASCII 0x61 through 0x7A,
0-9	ASCII 0x30 through 0x39,
dash ”-”	ASCII 0x2D), and
underscore ”_”	ASCII 0x5F

- not begin or end with a dash or underscore.

6C.2.2 Reserved directory names

The following directory names are reserved for specific purposes within PDS and may not be used elsewhere:

```
browse
calibration
context
data
document
geometry
miscellaneous
spice_kernels
xml_schema
```

6D Identifiers

An *identifier* is a character string that is uniquely associated with an object, product, collection, or bundle. Identifiers are strings of no more than 255 ASCII printable characters (see Section 6A.3); the sets of allowed characters depend on the type of identifier.

6D.1 Local Identifier

A *local identifier* is unique within a label. It can be assigned to distinct parts of the label to simplify cross-referencing within the label; for that reason, local identifiers should be easy for humans to read, understand, and trace. The characters must be from the ASCII alphanumeric character set, underscores, colons, and/or dashes as follows:

upper case letters	A-Z	ASCII 0x41 to 0x5a
lower case letters	a-z	ASCII 0x61 to 0x7a
digits	0-9	ASCII 0x30 to 0x39
dash	"-"	ASCII 0x2D, and
underscore	"_"	ASCII 0x5F

6D.2 Logical Identifier

A *logical identifier* (LID) is uniquely associated with the set of all versions of a bundle, collection, or product. It is constructed as four (bundle), five (collection), or six (product) fields from the following list of ASCII characters:

lower case letters	a-z	ASCII 0x61 to 0x7a
digits	0-9	ASCII 0x30 to 0x39
dash	"-"	ASCII 0x2D,
period	"."	ASCII 0x2E, and
underscore	"_"	ASCII 0x5F

```
urn:nasa:pds:<bundle_id>
urn:nasa:pds:<bundle_id>:<collection_id>
urn:nasa:pds:<bundle_id>:<collection_id>:<product_id>
```

where colons (ASCII 0x3A) delimit the fields (and may not be used within fields), and each field must begin with a letter or digit.

6D.3 Versioning

A *version identifier* (VID) may be appended to a logical identifier to select one of several versions of the same bundle, collection, or product. The combination is called a *versioned identifier* (LIDVID). LIDVIDs are used to locate products within PDS; every version of every product within PDS has a unique LIDVID.

VIDs are separated from LIDs by a double colon ("::") and have the form `M.n` where `M` and `n` are both integers:

```
urn:nasa:pds:<bundle_id>::<version_id>
urn:nasa:pds:<bundle_id>:<collection_id>::<version_id>
urn:nasa:pds:<bundle_id>:<collection_id>:<product_id>::<version_id>
```

Version identifiers are issued sequentially, although not all version numbers may appear in an archive. `M` denotes a major version and `n` denotes a minor version. `M` is initialized to "1" for archives, but "0" may be used for samples and tests; `n` is initialized to "0". Whenever `M` is incremented, `n` is reset to "0". Neither `M` nor `n` should be prepended with zeros; each is simply incremented as an integer. Thus "1.1" and "1.10" are different versions, and "1.01" is invalid.

When the contents of a collection are changed (because member products have been added or removed or because existing members identified by LIDVID have been replaced by newer versions),

the VID of that collection must be incremented. Similarly, a bundle VID must be incremented if any of its member collections change.

6E Classes, Attributes, and Attribute Values

6E.1 Classes

The following rules govern the selection of class names throughout PDS.

- Class names must be constructed from one or more components, each of which is made from members of the ASCII alphanumeric character set (see Section 6A.2).
- Class names must clearly convey meaning and pertinence (*e.g.*, Record_Binary).
- Components must be separated by ASCII underscores.
- Each component must begin with an upper case letter; all other letters must be lower case (*e.g.*, Product_Collection) except when the class name incorporates an acronym (*e.g.*, SPICE_Kernel).
- The most significant component must be placed first with subsequent components in decreasing order of importance (*e.g.*, Product_Table_Character rather than Character_Table_Product).
- Articles and prepositions must not be used as components
- Components must clearly convey meaning and pertinence (*e.g.*, Record_Binary).
- When practical, subclass names must include part of the parent class name (*e.g.*, Array_2D_Image is a subclass of Array_2D, which is a subclass of Array)
- The class name must not exceed 255 characters.

6E.2 Attributes

The following rules govern selection of attribute names throughout PDS.

- Attribute names must be constructed from one or more components, each of which is made from lower-case ASCII letters and/or digits
- Attribute names must begin with a lower-case ASCII letter.
- Attribute names must clearly convey meaning and pertinence (*e.g.*, institution_name).
- Components must be separated by ASCII underscores.
- The least significant component must be placed first with subsequent components in increasing order of importance (*e.g.*, maximum_value).
- Articles and prepositions must not be used as components
- Components must clearly convey meaning and pertinence (*e.g.*, institution_name).
- The attribute name must not exceed 255 characters.

6E.3 Attribute Values

Attribute values should be capitalized as in normal usage — lower case except for acronyms, the first character in proper names, the first character at the beginning of a sentence, etc. Single words or short phrases should generally be lower case. Underscores should not be used as separators unless they would appear in usage outside labels. With only rare exceptions (*e.g.*,

enumerated values or at the request of a peer review), PDS will not reject attribute values that fail to meet these guidelines; the guidelines are primarily to help humans read label information.

PDS searches on attribute values will be case sensitive for enumerated values and case insensitive otherwise.

6E.3.1 Attribute Value Units

Attributes that are measurable quantities — *e.g.*, <altitude>, <file_size>, and <latitude> — must have the unit of measurement specified by an XML attribute 'unit' in the tag. The unit must be selected from the set of possible units given by <unit_id> in the appropriate Unit_Of_Measure subclass. For example:

```
<altitude unit="m">173.293</altitude>
```

where the possible choices from Unit_of_Measure_Length are:

```
Angstrom   nm   micrometer   mm   cm   m   AU
```

6E.3.2 Attribute Value Limits

Attributes may be limited to a specific set of values if specified in the defining data dictionary. Specific values are set (and their meanings defined) by repeated use of the DD_Permissible_Value class in the data dictionary definition. For example, the attribute <encoding_type> may have the values `binary` or `character` depending on the class in which it is used.

Limits on the range of attribute values (and/or maximum or minimum numbers of characters in character strings) may be imposed by using the meta-attributes <maximum_value> and <minimum_value> (or <maximum_characters> and <minimum_characters>) in the DD_Value_Domain class. Units for permissible values, <maximum_value>, and <minimum_value> are set by meta-attribute <specified_unit_id> in the data dictionary, where the <specified_unit_id> must be selected from the possible values given by <unit_id>.

7 Units

Uniform use of units facilitates catalog searches across archive systems. The PDS standard for units is the *Systeme Internationale d'Unites* (SI); see http://physics.nist.gov/cuu/Units/units.html for details.

There are two levels at which units are specified — the type of measurement (*e.g.*, length, mass, time, etc.) and the specific units in which a measurement is reported (*e.g.*, meters, kilograms, seconds, etc.) — and three applications (data dictionaries, attribute values in labels, and data objects).

7A Types of Measurement

When an attribute denoting a measurable quantity is added to a PDS data dictionary, the type of measurement must be specified using the <unit_of_measure_type> meta-attribute. Valid choices (among many others) include the classes

Unit_Of_Measure_Length
Unit_Of_Measure_Mass
Unit_Of_Measure_Time

The definition of each class includes the attributes <type> (the last field in the class name—*e.g.*, length, mass, time, etc.), <unit_id> (the set of acceptable abbreviated units for use in labels), and <specified_unit_id> (the unit in which meta-attributes <maximum_value>, <minimum_value>, and <value> are given within the `DD_Value_Domain_Full` specification). Examples are shown in the table below:

type	unit_id
acceleration	cm/s**2 km/s**2 m/s**2
angle	arcmin arcsec deg hr mrad rad
length	AU Angstrom cm km m micrometer mm nm

7B Specific Units

For attributes denoting measurable quantities, the specific unit must be selected in the label using an XML attribute; the XML attribute <unit> identifies the selected unit from the possibilities provided by <unit_id> in the data dictionary.

```
<length unit="km">3</length>
```

In the XML statement above, <length> is the attribute, "unit" is the qualifying XML attribute, and "km" is its value — the specific unit. That is, the value of <length> is 3 km.

7C Prefixes

Prefixes (or their abbreviations) shown in the table below may be prepended to any SI unit.

SI PREFIXES

Factor	Prefix	Symbol	Factor	Prefix	Symbol
10**24	yotta	Y	10**-1	deci	d
10**21	zetta	Z	10**-2	centi	c
10**18	exa	E	10**-3	milli	m
10**15	peta	P	10**-6	micro	
10**12	tera	T	10**-9	nano	n
10**9	giga	G	10**-12	pico	p
10**6	mega	M	10**-15	femto	f
10**3	kilo	k	10**-18	atto	a
10**2	hecto	h	10**-21	zepto	z
10**1	deka	da	10**-24	yocto	y

There is presently no ASCII symbol for “micro”.

7D Derived Units

If the needed specific unit is not available from the PDS data dictionary, PDS allows construction of a new unit by concatenating units and prefixes that have been approved and common mathematical symbols (“-“ for negation, “**” for exponentiation, “*” for multiplication, and “/” for division in that order of precedence). In this way a data provider can express flux in watts per square meter (w/m**2), for example. Note that map scales and radiance, examples of other parameters using compound units, are already in the PDS data dictionary.

7E Expressions

Units are written in lower case except for those derived from proper names or acronyms or which have upper case symbol prefixes. No periods are used in any of the abbreviations.

Expressions should be in the simplest (shortest) form. For example, although “mkm” and “m” are numerically identical, the first is not allowed; and “w/m**2” is preferred over “w*m**-2”.

8 Content

PDS requires that products, collections, and bundles be documented so that scientists in future years can understand (1) how the data were collected and processed, (2) what the data mean, and (3) the limitations of the data. These future scientists are assumed to have backgrounds comparable to the people who archived the data; they should be able to assess the value of the data in new work and then use the data. This section summarizes *what* should be included in PDS archives as opposed to how it is presented, which was the focus of Sections 1-7; the emphasis is on documentation.

8A Documentation Requirements

Documentation takes three forms in PDS: (a) XML labels, (b)documents included within the archive, and (c) references to material that is publicly available elsewhere.

Data providers must negotiate with the PDS consulting node early in the archiving process on the documentation requirements and how they will be distributed among the three categories above. Peer review provides feedback on whether the goals have been achieved. Questions to be asked at each step include:

- What is necessary for evaluation, understanding, and use of the data?
- What might be useful beyond the 'necessity' threshold?
- What is publicly available?
- How should documentation be divided among labels, internal documents, and external documents?

The contents of the documentation must include (but are not limited to) detailed descriptions of the instrument, spacecraft or other instrument 'host', investigation(s), data acquisition procedures, and analysis procedures, facilities, and personnel. If additional processing is anticipated, those steps should be outlined.

8A.1 Labels

Labels provide machine-readable information about structure and relationships. The label structure is discussed in more detail elsewhere (see Section 3).

Labels also provide references. This is the primary mechanism by which the relationships are established among products within and external to the archive. An Internal Reference is used to establish references to products (*e.g.*, document products, context products, other collection products, etc) that are within the PDS repository. An External Reference is used to establish references to products that are available external to the PDS (*e.g.*, material that is publicly available elsewhere).

8A.2 Internal Documentation

Internal documentation is one or more document products that have been included in a bundle. As with other products, each document product must be a primary member of one collection (and may be a secondary member of many). Most document products will be members of a document collection, which is then a member of a bundle; however, some documents may appear elsewhere, such as in a calibration or miscellaneous collection. Bundles are not required to have document collections, but they may have several. It would be unusual for a bundle not to include at least one document collection.

8A.2.1 Document Collections

Document collections must be specified by a Product_Collection, which consists of an inventory table and an XML label.

The data provider will work with the consulting node on partitioning documents among collections. For most bundles, a single Document Collection, possibly organized with multiple subdirectories, will be sufficient. In other cases, it may be preferable to separate documents of different types into a few collections. For especially complex documents, it may be most practical to assign all the pieces (text, figures, tables, etc.) to a single document collection — in which case the result may be one document per collection. In other cases, it may be sufficient to separate documents of different types into a few collections.

Document products related to calibration may be primary members of a calibration collection rather than a document collection. Documents related to 'geometry' may be primary members of a geometry or SPICE kernels collection rather than a document collection.

8A.2.2 Document Products

The range of relevant documents will depend on the type(s) of data and the scope of the associated collections and bundle. In general, PDS seeks to err on the side of completeness. For a typical planetary spacecraft mission, documents could include:

- Mission Plan
- Instrument papers
- Instrument user guides
- Calibration reports
- Science articles
- Software Interface Specifications (SISs)
- Software user manuals

Including a document in a PDS archive constitutes publication (or re-publication) of that document. Consequently, documents must meet not only the PDS label and format requirements, but also the structural, grammatical and lexical requirements of a refereed journal. Documents which contain spelling errors, poor grammar, or illogical organization will be rejected and may ultimately lead to rejection of the submitted data.

It is the responsibility of the data provider to obtain clearance for materials judged to be sensitive under U.S. International Traffic in Arms Regulations (ITAR) and to obtain permission from copyright holders for re-use of protected materials in the archive. PDS reserves the right to request documentation of either or both ITAR and copyright permissions.

Each document must be saved in one or more PDS-compliant formats using the Product_Document class. A 'human-readable' and/or PDF/A version of each file is required; versions in other approved formats are optional. All formats are described by the same Product_Document.

For documents, there can be only one data object per file.

A file is 'human-readable' if it is not encoded and if any special markup tags which may be included do not significantly interfere with an average user's ability to read the file. For example, simple HTML files and TeX/LaTeX files with relatively little markup embedded in the text are generally considered human-readable and may be used to satisfy the requirement.

Flat ASCII text is another option for the human-readable format. Flat ASCII text means that the file contains only the ASCII printable character set (see Section 6A.3) plus the carriage-return and line-feed pairs as the record delimiter.

8A.3 External Documentation

For archive integrity, it is preferable to have all relevant documentation within the archive; in many cases, however, this is either impractical or impossible. For example, when a copyright holder refuses re-use, the copyrighted material must be referenced through an outside source.

The Reference_List class may be used to point toward relevant material either inside or outside PDS; it may be associated with any of over two dozen different product classes including Product_Context, Product_Documentation, Product_Collection, and Product_Browse.

The External_Reference class provides a complete bibliographic citation to a published work outside the archive, optionally including its Digital Object Identifier (DOI). External_Reference may be used through Reference_List or directly, such as from Observing_System_Component. External_References to inaccessible materials may lead to rejection during peer review.

8B Context Bundle, Collections, and Products

"Context" products are terse summaries of (and links to) other documentation that can help users navigate within PDS; they contain nothing that is essential to understanding and using PDS data that cannot be found elsewhere. However, they can represent a simple source for associating references to key physical and conceptual objects. Consequently, while most context products are prepared by data providers, they become members of Collections for which the PDS Engineering Node is the steward. Those collections are members of a single PDS 'master' context bundle which is used widely across the system.

8B.1 Collections Under the PDS4 Context Bundle

The PDS 'master' context bundle contains separate 'master' context collections, one for each type of context product (*e.g.*, instrument, telescope, facility, PDS Node, etc.).

8B.2 PDS Context Products

A physical or conceptual object referenced by a PDS product must have its own context product; for example, every spacecraft will have a context product and each instrument on the spacecraft will have a context product. Every context product must be a primary member of one of the collections in the master context bundle.

A data provider — working with a consulting node — is generally responsible for providing the content for a context product. However, since the PDS Engineering Node is the steward for the entire master context bundle, the provider must also work closely with EN personnel on product construction, starting with LID and VID assignments for each new context product.

A context product is a single XML label for the physical or conceptual object. The contents of the label are simple XML statements that give the logical identifier (LID) of the product, the name and abbreviation (if any) of the object, the type of object (*e.g.*, instrument, node, target, etc.), a brief description of the object suitable for display within a web interface, and reference associations for the critical documents describing the object.

8B.3 Context Collections and Products in Science Bundles

A data provider may optionally include a context collection when submitting a bundle containing scientific data to PDS. That context collection may be either a primary or secondary member of the bundle. The collection's own membership is specified in its Product_Collection – a collection inventory table and its associated XML label.

The context products associated with the collection — even those shepherded to completion by the data provider — must be secondary members of the science bundle's context collection. The submission details (delivery method and timing) for new context products are negotiated among the provider, the consulting node, and EN. As noted in Section 2A.4, secondary members of a context collection need not be included in deliveries to the PDS.

When science bundles do not include context collections, references must be to context products and collections in the PDS4 Context Bundle.

8C Calibration

Instruments are imperfect sensors; their outputs require adjustment so that the measurements are quantitatively meaningful, presented in physical units, and as accurate as possible. PDS requires that raw data be included in archives and that the calibration steps subsequently applied be documented. The details will vary among instruments types and disciplines; data providers should negotiate archiving requirements with consulting nodes. Calibration products would

ordinarily be members of a calibration collection (see Section 2A.5), but other options are possible — such as when calibration data are interleaved with science observations.

8D Geometry

Planetary flight missions are expected to archive complete geometric details from launch through end of mission. These typically include full ephemerides of the spacecraft and relevant planetary bodies, orientation of the spacecraft and all instruments, the relationship of these to coordinate systems on the target(s), a history of all significant spacecraft events, and other 'housekeeping' data (such as temperatures and power levels) that might be useful in understanding the behavior of instruments. Radiometric tracking data should be archived even if there is no 'instrument team' for radio science.

The archives of Earth-based observing campaigns and other observations must include equivalent geometrical data. The objective in each case is to allow future data users to reconstruct the observing geometry in a way that supports quantitative analysis.

Geometry products should be members of a geometry collection or a SPICE kernels collection; the distinction is described in Section 2A.5.