Data Providers' Handbook

Archiving Guide to the PDS4 Data Standards

DRAFT

Data Design Working Group

31 October 2011

Version 0.3.5

CHANGE LOG

Revision	Date	Description	Author
0.1	Mar 30, 2009	Initial draft based on information collected by the Data System Working Group.	R.Joyner
0.2	Aug 6, 2009	Updated versions of all Classes	R. Joyner
0.2.1	2010-08-31	Complete overhaul, but only partly successful through page 25	R. Simpson
0.22	Aug 31, 2010	Integrate Simpson and Joyner docs	R. Joyner etal
0.22.1	2010-09-22	Edits through Section 3, except Section 2	Simpson
0.22.2	2010-09-24	Added comments from Mitch Gordon	Simpson
0.22.2	2010-10-01	Significant edits to Section 4	Simpson
0.22.3	2010-10-25	Significant edits to Sections 1,3-6	Simpson
0.3	2011-04-15	Significant edits addressing Build 1b comments	M.Gordon etal
0.3.1	2011-04-21	Additional content added	M.Gordon etal
0.3.2	2011-05-02	Significant reorganization, additional edits	M.Gordon etal
0.3.3	2011-06-29	Make Sections 1-2 more user friendly	R. Simpson, M. Gordon
0.3.4	2011-09-06	Major changes Sections 2, 3, 8,14	M. Gordon, R. Joyner
0.3.5	2011-10-31	Updates for schema 0.5.00g	M. Gordon, R. Joyner

TABLE OF CONTENTS

1.0 INTRODUCTION

Planetary Data System version 4 (PDS4) represents a departure from previous versions of the PDS. Although it is still an archive of planetary data, it has been designed using contemporary information technology concepts and tools. The system is built around a 'data model' that rigorously defines each of its components and the relationships among them. There are only four fundamental data structures, but many extensions are possible — each also rigorously defined. By carefully controlling product definitions and relationships, PDS can accurately track the progress of each product entering the system, compute detailed inventories of holdings, design sophisticated services that users can request to act on subsets of the archive (such as transformations and displays, in addition to the expected search and retrieval functions), and connect data products to relevant internal and external information (documentation).

NOTE: Some example XML fragments and data dictionary exerpts have not been updated to the most recent versions of the source material. Consequently, some entries may not match exactly the current schema or dictionary; however the process and considerations being discussed should be valid.

1.1 Purpose

The *Data Providers Handbook* (*DPH*) is a guide for preparation of data being submitted to PDS4. It will walk you through preparation of very simple products, collections of products, and bundles of collections, which are the units in which deliveries are made to PDS4.

1.2 Audience

The *DPH* is written for scientists and engineers in the planetary science community who are planning to submit new or restored data to PDS4 (data providers). While the document is applicable to all such submissions, most of the examples and discussions are presented in a mission/instrument context.

1.3 Reader Preparation

The *DPH* is one of several documents describing the new system.

Readers, even those very familiar with previous versions of PDS, **should read the *PDS4 Data Standards Concepts Document* [4] and the *Glossary of PDS4 Terms* [5] before beginning the Data Provider's Handbook.**

The *DPH* should be used in conjunction with the *PDS Standards Reference* (PDSSR) [2] and the *PDS4 Data Dictionary* [3a,b], which have been updated for PDS4. For more information on the individual documents, see the *Introduction to the PDS4 Document Set* [7].

1.4 XML Editors

PDS4 is implemented in the eXtensible Markup Language (XML), a set of 'open source' rules for encoding documents and data structures in machine-readable form with special applicability to providing web services. It is beyond the scope of the *DPH* to include an XML tutorial; however, Appendix B "Reading a PDS4 XML Schema" provides some introductory information.

The use of an XML editor simplifies construction and validation of schemata (the plural of 'schema') and labels. Two editors have been popular during development of PDS4.

oXygen	**http://www.oxygenxml.com**	which is licensed software, and
Eclipse	**http://www.eclipse.org/**	which is open source software.

1.5 Document Notation

A few words have meanings which differ depending on the community in which they are used. We have adopted modifiers to help distinguish among multiple uses. For example, 'attribute' is widely used in both PDS and XML — but its meaning in each case is different. In this document we use 'PDS attribute' and 'XML attribute' to establish the context.

1.6 Applicable Documents

1.6.1 Controlling Document

[1] Planetary Data System (PDS) PDS4 Information Model Specification, Version 0.5.0.0.g

1.6.2 Reference Documents

[2] Planetary Data System Standards Reference, JPL D-7669, Part 2

[3a] PDS4 Data Dictionary - Abridged - V.0.5.0.0.g.

[3b] PDS4 Data Dictionary - Unabridged - V.0.5.0.0.g.

[4] PDS4 Data Standards Concepts, September 12, 2011.

[5] Glossary of PDS4 Terms, October, 2011, Version 2011-10-28 (v111028).

[6] PDS4 Data Dictionary Tutorial, October 28, 2011.

[7] Introduction to the PDS4 Document Set, October 30, 2011.

1.7 Other Resources

1.7.1 XML Schema location

http://pds.nasa.gov/schema/pds4/current/generic/common/

1.7.2 PDS4 Software

http://pds.nasa.gov/schema/pds4/current/generic/common/ (bottom of the page)

[TBD need a permanent link]

Part I. Preliminaries

2.0 BUILDING BLOCKS

2.1 Terminology

The results of our exploration of the Solar System can be loosely described as *data objects*; these can be electronic files, dust samples, or a sense of awe at the wonder of the universe. For purposes of archiving, we need a *description* to accompany each data object — in the case of an electronic file, a digital object, we need both the structure and meaning of the file contents for it to be useful. We can't fit dust samples or senses of awe into PDS4, but we can fit their descriptions. A description paired with its data object (when available — *e.g.*, ifa digital object) is called an *information object*. If many data objects have similar characteristics, we can group them into a *class* and the common characteristics are the defining *attributes* of that class.

A *product* is one or more closely related information objects for which the descriptions have been combined into a single XML *label* and for which the product has a PDS-unique *identifier*. Closely related products may be grouped into a *collection*; in fact, every product entering PDS must be a member of some collection. Closely related collections may be grouped in a *bundle*.

For example, a planetary image, the histogram of its pixel values, and the descriptions of both could be organized as a product. Many such products — perhaps of the same target — could be defined as a collection. Image collections from many targets along with appropriate documentation, calibration, etc. (separate collections) could be a bundle, which would be a deliverable to PDS.

For more rigorous definitions of the terms introduced above, see the *Glossary of PDS4 Terms* [5].

2.2 Data

A word of caution about terminology – we have tried to avoid using differently terms that have strong PDS3 connotations. Unfortunately the English language does not provide a sufficient set of meaningful, unique, unambiguous terms to meet all of our needs. Please do not rely on the names of things – review carefully the PDS4 definitions.

Recall from the PDS4 Glossary, a “Product” is one or more closely related information objects grouped together and having a single PDS-unique identifier. In the PDS4 implementation, the descriptions for all of the objects making the product are combined into a single XML label.

Four basic structural data formats are allowed in PDS4.

1. repeating_record_structure
    - Suitable for fixed length tabular data.

  - May be either binary or character, but a single object may be only one of these.
  - The records are fixed length.
  - The elements of a record (fields) are heterogeneous.
  - The formats (data type and size) of fields in the same position in each record are the same (i.e., the second field in the second record is constructed identically to the second field in the first record).
2. homogeneous_array_structure
   - Suitable for images, spectra, spectral cubes, maps, etc.
   - The elements of an array are homogeneous.
   - The individual elements of any array are stored with their bytes in the order dictated by their scalar type.
   - PDS requires a specific order in which the elements of the array are stored. The order is described in the Concepts Document, Section 9.3, and the Standards Reference, Section 5.2.

The majority of PDS4 objects can be supported by the two previous structures. For those PDS4 objects which can not supported by the above, we have two additional stuctures distinguished by whether or not software must be used to decode the information before it can be accessed for reading, display, or analysis.

3. parsable_structure
   - Suitable for plain text, HTML, XML, tabular data with variable length fields and records (delimited text).
   - The contents are a byte stream which can be parsed with standard rules (e.g., comma separated entries, standard punctuation);t no decoding software (e.g., Adobe Acrobat©) is required.

4. encoded_structure
   - Suitable for documents, browse products, etc., but not generally not for observational products.
   - Contents are a byte stream that must be decoded by software before use.
   - The use of encoded_byte_stream objects is restricted by PDS to a limited set of PDS approved external standards (e.g., PDF-A, JPEG, GIF).
   - Only in exceptional cases will encoded_byte_stream objects be considered appropriate for storing observational data. Prior PDS approval is required.

Each of these structural formats corresponds to one PDS4 "base class" and each PDS4 "base class" uses one and only one of the structural formats.

Structural Format		Base Class
• repeating_record_structure	↔	Table_Base
• homogeneous_array_structure	↔	Array_Base
• parsable_structure	↔	Parsable_Byte_Stream_Base
• encoded_structure	↔	Encoded_Byte_Stream_Base

Here are a few rules (for a full description, see the Standards Reference, Section 5):

- Each digital object must be stored in one of the four basic data formats.
- A digital object must be contained in a single file.
- A file may contain multiple digital objects from the same product.
- Digital objects within a file are not required to use the same storage structure.
    - Can have a header (parsable structure) and an image (array structure) in the same file.
- When multiple digital objects are contained in a file, they must be stored sequentially, i.e., readout of the file should be one object followed by the next, not two objects interleaved.

These data structures only tell you how to read the bytes from a file; they contain absolutely no interpretation beyond that. PDS4 defines a set of base object classes that apply the first level of interpretation to the bytes read (the "It's an image" interpretation), and subclass extensions of those base object classes to expand and restrict the associated metadata for specific object types.

2.2.1 Sample Data

This document frequently references a set of example PDS4 products – see Section 16 Example PDS4 Products.

There are two different sets of examples:

- The first is a set of example products. This includes a representative set of products that would exist within an archive (e.g., character table, binary table, document, etc.).

- The second set is a example complete archive. This includes all products that would comprise a complete PDS4 archive (e.g., bundle, collections, collection inventories, aareadme, errata, and basic products).

2.2.2 Methodology

To complete the sample data (from above), we created XML labels for each of the products in the archive. For simplicity, we assumed that each product is an object-description pair. The products are grouped into collections. The collections are grouped into a single archive bundle, and the bundle was delivered to PDS.

3.0 SCHEMA AND LABELS – AN OVERVIEW

Label development begins with an XML schema. PDS maintains a library of generic XML schema for each product type based on the core portion of the PDS information model.

In consultation with your PDS discipline node (DN), for each product type, determine which schema you will need. The DN will take the appropriate generic schema and do some initial editing before passing a 'tailored' version of the schema to you. The node's editing will probably include changing some classes and attributes from' optional' to 'required', and inserting node and possibly mission specific classes appropriate for the products you plan to produce.

As the data provider, you will modify the tailored schema to produce a 'specific schema', one that is designed for your data. Your specific schema must be validated against the tailored schema you received from the DN which will have been validated against the generic schema on which it is based.

Finished labels must be validated against the specific schema, tailored schema and generic schema on which they are based.

3.1 Pipeline Considerations

If you are developing a collection with more than a few products, you will want to automate label generation as much as possible.

There are at least two approaches:

a) Use the specific schema (xsd file) as input to the pipeline software you use to generate your labels, or

b) Use a label template (xml file) as input to the pipeline software. One feature of many XML editors is the ability to generate such a template .

The template looks like the final label except that, depending on how you set some options, there are either no values between the XML tags, or the values which vary from one label to the next are represented by placeholders which the pipeline software will replace.

As with everything, there are advantages and disadvantages to each approach. The first consideration will probably be the software package underlying your pipeline and whether or not it is specifically designed to handle XML.

In principle, you should make as many of your edits as possible in the specific schema to provide yourself and PDS the most robust criteria against which to validate the individual XML label files produced for your archive.

In this handbook, we assume the pipeline will use an XML template. This provides a framework for the discussion. Even if your pipeline will use the specific XML schema as input, you will still need to read the sections on 'Schema Editing' including the section on 'Label Template Editing'.

In this handbook, we use Product_Table_Character for most of our examples. The tailored schema you receive from your node may differ slightly from the ones in these examples. Our goal here is to become familiar with the contents and to understand the process.

3.2 Permitted Schema Modifications

When you modify a parent schema, the resulting schema must still satisfy its parent schema's restrictions. In principle this means you can modify the schema to make it more restrictive, and you may also add classes within certain constraints.

a) You may change an XML element from optional (minOccurs=”0”) to required (minOccurs=”1”, or some number that is less than or equal to the value specified in maxOccurs).
b) You may delete an optional XML element (if minOccurs=”0”)
c) You may restrict the upper limit on the number of times the XML element will be used by setting maxOccurs to a value that is greater than or equal to the value specified in minOccurs.
d) For those XML elements which have “maxOccurs = unbounded”, you may set an explicit upper limit on the number of times the XML element will be used.
e) If an XML element will have the same value for all products being produced from this schema you may insert that value into the XML element in the schema.
f) During the initial tailoring, the PDS node staff may insert additional classes in either the Mission Area and / or the Node Area (subsections within the Observation_Area).
g) The data provider, may insert additional classes in the Mission Area (a subsection within the Observation_Area).

You may not modify the minOccurs or maxOccurs attributes such that they would specify a less restrictive set of conditions. For example, if “minOccurs = 5” you cannot set “minOccurs = 4”, as this would violate the initial restriction. Similarly, if “maxOccurs = 1”, you cannot set “maxOccurs = 2”.

There are really three options for what you will do in your schema with XML elements which are 'optional' in the parent schema:

1. The XML element will be used across all labels
2. The XML element will sometimes be used in one or more labels.
3. The XML element will not be used in any labels

There are several approaches to handling optional XML elements based at least in part on your label pipeline design. We recommend:

- Set “minOccurs = 1” (or some appropriate higher value) for the optional XML elements you intend to use in every label.
- Delete the optional XML elements that will not be used in any label (if “minOccurs = 0”).
- Leave “minOccurs = 0” for the optional XML elements you intend to use only for some subset of the products. When maxOccurs is unbounded, reset it to an appropriate upper value. Note maxOccurs must be greater than or equal to minOccurs.

3.3 Developing Local Data Dictionaries

Local data dictionaries are developed by the data preparer in conjunction with discussions and review by the lead PDS discipline node. Local dictionaries contain new classes and attributes as needed to provide detailed product descriptions which can not be given with just the existing existing PDS classes and attributes

A data preparer creates a local dictionary when he needs to define attributes and classes specific to his mission, observing campaign, data restoration effort, etc, including but not limited to specific instrument parameters, and additional observational parameters.. They are an essential tool for providing data preparer’s latitude to tailor their labels to more closely fit the particular nature of the observational and supplementary data in the archive.

Local data dictionaries are discussed in more detail in Section 13.

4.0 USING THE DATA DICTIONARY

The bulk of this handbook is concerned with developing and producing PDS4 compliant labels. You will frequently need to refer to the PDS4 Data Dictionary and possibly the PDS4 Data Dictionary Tutorial.

For the purposes of our discussion and the examples that follow, we will be using the Abridged version of the dictionary.

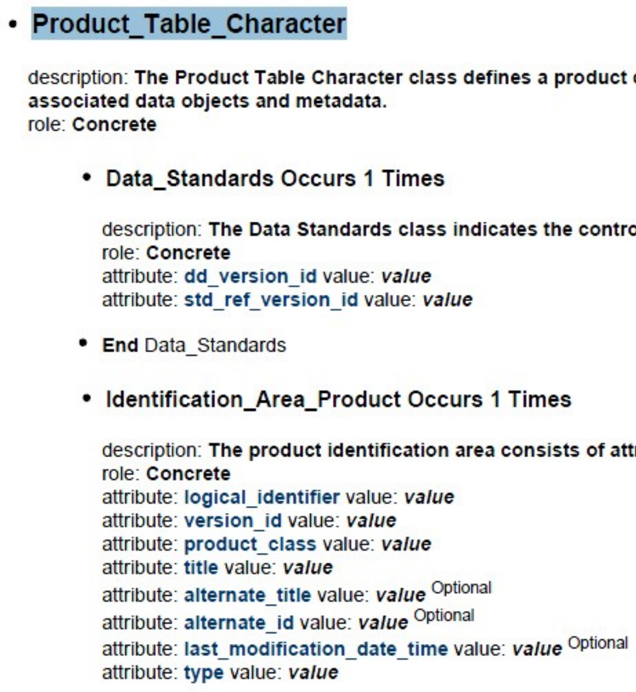
Even if you are only interested in a particular class or attribute, we recommend the following strategy.

a. Search the dictionary for the product type you are working on. For example, search for Product_Table_Character if you are working on a character table schema or label.
b. Select the search result with the full description of the product.
c. Scroll through that entry for the appropriate block in your schema or label.
d. Find the class or attribute in question.
e. Follow hyperlinks as necessary to get the information you need.

4.1 Data Dictionary Example: last_modification_date_time

This section demonstrates one of the many ways in which to use the data dictionary. The discussion that follows demonstrates, at a high level, how to interpret the dictionary and the implications of the restrictions specified in the dictionary on populating attribute values.

First, have the dictionary open to the ‘Product_Table_Character’ section:



- **Product_Table_Character**

  description: **The Product Table Character class defines a product consisting of at least one character table and other associated data objects and metadata.**
  role: **Concrete**

  - **Data_Standards Occurs 1 Times**

    description: **The Data Standards class indicates the controlling standards for this product.**
    role: **Concrete**
    attribute: **dd_version_id** value: ***value***
    attribute: **std_ref_version_id** value: ***value***

  - **End** Data_Standards
  - **Identification_Area_Product Occurs 1 Times**

    description: **The product identification area consists of attributes that identify and name a data product.**
    role: **Concrete**
    attribute: **logical_identifier** value: ***value***
    attribute: **version_id** value: ***value***
    attribute: **product_class** value: ***value***
    attribute: **title** value: ***value***
    attribute: **alternate_title** value: ***value*** Optional
    attribute: **alternate_id** value: ***value*** Optional
    attribute: **last_modification_date_time** value: ***value*** Optional
    attribute: **type** value: ***value***

From the above, we can see that Product_Table_Character has a Data_Standards class that occurs once and a Identification_Area_Product class that occurs once. Each of the sub-classes has a description / definition and a number of attributes.

We are interested in knowing more about the 'last_modification_date_time' attribute in the Identification_Area_Product class. We can click on:

attribute:**last_modification_date_time**

Here is the full entry the link takes us to:

last_modification_date_time
steward: **pds**
name space id: **pds:**
version: **0.4.1.1.f**

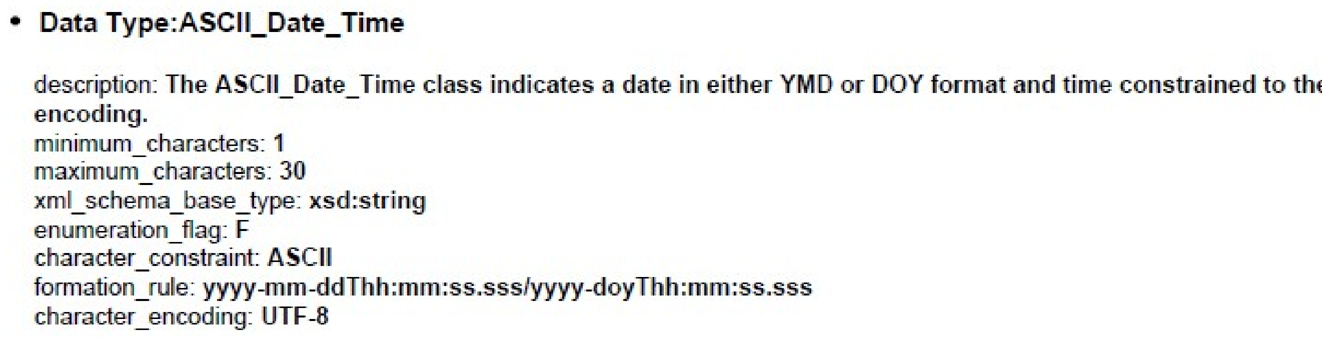
- description: **The last modification date time attribute gives the most recent date and time that a change was made.**
- data_type: **ASCII_Date_Time**

We only need the description which tells us what this particular attribute means, and the data_type which tells us the formation rules. You can learn what the other entries mean by reviewing the terminology section of the Data Dictionary and the PDS4 DD Tutorial Document.

We know that since this attribute is in the Identification_Area, the attribute will occur in pretty much every product in our archive.

The value we should use is the last time either the label or the data file was modified. We are going to assume that the label will be updated later than the data file (that the label describes). So, we will populate the 'last_modification_date_time' attribute with the date/time when the label file was written (by our pipeline

In the data dictionary the value for data_type, ASCII_Date_Time, is a hyperlink which leads to:



- **Data Type:ASCII_Date_Time**

description: **The ASCII_Date_Time class indicates a date in either YMD or DOY format and time constrained to the**
encoding.
minimum_characters: **1**
maximum_characters: **30**
xml_schema_base_type: **xsd:string**
enumeration_flag: **F**
character_constraint: **ASCII**
formation_rule: **yyyy-mm-ddThh:mm:ss.sss/yyyy-doyThh:mm:ss.sss**
character_encoding: **UTF-8**

The description tells us we can give the date as either year-month-day or as day-of-year, and that we are constrained to the ASCII portion of UTF-8 character encoding.

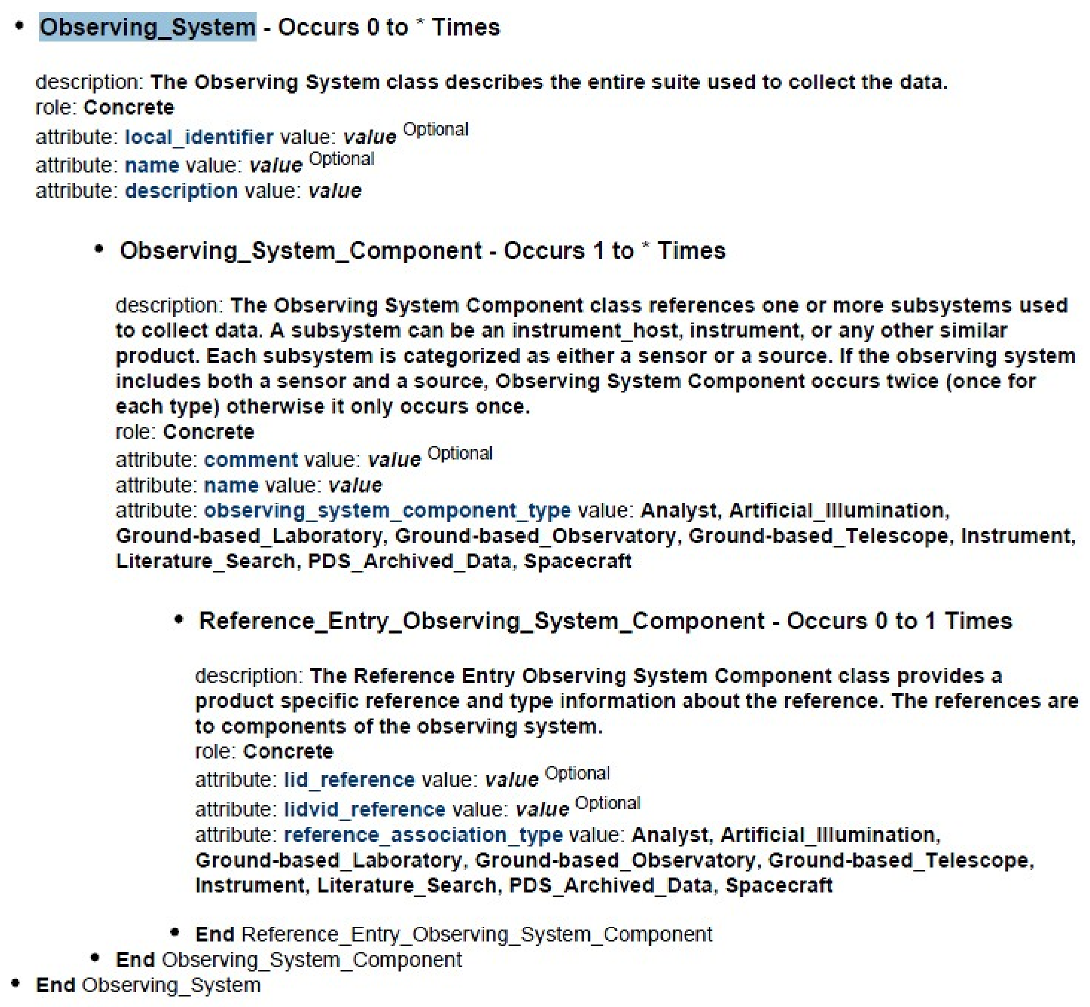
The formation rule is fairly clear, but notice the number of character restrictions. Minimum_characters:1 (This probably will be changed to 4 in the next version of the dictionary). The implication is that the value can be truncated to specifying, at a minimum, the 4-digit year. At the other extreme, we probably do not need to preserve the time the product was last modified to the nearest millisecond.

In our examples, we use YMD format with time to the nearest second which is still probably unnecessary precision, but we wanted to show an example of the date-time format.

4.2 Data Dictionary Example: Observing_System

This example illustrates parallels between schema content and structure and dictionary content and structure, and how to determine what values are suitable or permitted for a particular entry. Observing system gives a nice small example of nested classes within a schema.

Again, we have the dictionary open to the 'Product_Table_Character' section, and scroll down to the Observing_System subclass within the Cross_Reference_Area_Product section:



- **Observing_System - Occurs 0 to * Times**

  description: **The Observing System class describes the entire suite used to collect the data.**
  role: **Concrete**
  attribute: **local_identifier** value: ***value*** Optional
  attribute: **name** value: ***value*** Optional
  attribute: **description** value: ***value***

  - **Observing_System_Component - Occurs 1 to * Times**

    description: **The Observing System Component class references one or more subsystems used to collect data. A subsystem can be an instrument_host, instrument, or any other similar product. Each subsystem is categorized as either a sensor or a source. If the observing system includes both a sensor and a source, Observing System Component occurs twice (once for each type) otherwise it only occurs once.**
    role: **Concrete**
    attribute: **comment** value: ***value*** Optional
    attribute: **name** value: ***value***
    attribute: **observing_system_component_type** value: **Analyst, Artificial_Illumination, Ground-based_Laboratory, Ground-based_Observatory, Ground-based_Telescope, Instrument, Literature_Search, PDS_Archived_Data, Spacecraft**

    - **Reference_Entry_Observing_System_Component - Occurs 0 to 1 Times**

      description: **The Reference Entry Observing System Component class provides a product specific reference and type information about the reference. The references are to components of the observing system.**
      role: **Concrete**
      attribute: **lid_reference** value: ***value*** Optional
      attribute: **lidvid_reference** value: ***value*** Optional
      attribute: **reference_association_type** value: **Analyst, Artificial_Illumination, Ground-based_Laboratory, Ground-based_Observatory, Ground-based_Telescope, Instrument, Literature_Search, PDS_Archived_Data, Spacecraft**

    - **End** Reference_Entry_Observing_System_Component
  - **End** Observing_System_Component
- **End** Observing_System

Notice the nesting. Observing_System contains

1. a description,
2. a role,
3. three attributes, and
4. a subclass, Observing_System_Component.

   Observing_System_Component contains

   1. a description,
   2. a role,
   3. one attribute, and
   4. a subclass, Observing_System_Reference_Entry.

      Observing_System_Reference_Entry contains

      1. a description,
      2. a role, and

3. three attributes.

Let’s look at a slimmed down version of the dictionary entry retaining only the classes and attributes and compare it to an abbreviated extract from the schema.

Observing_System
attribute: value: **local_identifier *value*** Optional
attribute: value: **title *value***
attribute: value: **description *value*** Optional
    Observing_System_Component
    attribute: value: **observing_system_component_type SENSOR, SOURCE**
        Observing_System_Reference_Entry
        attribute: value: **lid_reference *value*** Optional
        attribute: value: **lidvid_reference *value*** Optional
        attribute: value: **reference_association_type has_association, has_instrument, has_instrument_host**
        End Observing_System_Reference_Entry
    End Observing_System_Component
End Observing_System

```
<xsd:complexType name="Observing_System_Type">
    <xsd:sequence>
      <xsd:element name="local_identifier" ...
      <xsd:element name="title" ...
      <xsd:element name="description" ...
      <xsd:element name="Observing_System_Component" ...
    </xsd:sequence>
</xsd:complexType>
```

The first three XML elements correspond to the first three attributes in the dictionary entry. The forth XML element corresponds to the class Observing_System_Component. The next section of the schema is the schema representation of the Observing_System_Component class.

```
<xsd:complexType name="Observing_System_Component_Type">
  <xsd:sequence>
    <xsd:element name="observing_system_component_type" ...
    <xsd:element name="Observing_System_Reference_Entry" ...
  </xsd:sequence>
</xsd:complexType>
```

This is followed by the schema representation of the Observing_System_Reference_Entry class.

```
<xsd:complexType name="Observing_System_Reference_Entry_Type">
  <xsd:sequence>
    <xsd:element name="lid_reference" ...
    <xsd:element name="lidvid_reference" ...
    <xsd:element name="reference_association_type" ...
  </xsd:sequence>
</xsd:complexType>
```

Much of the information in the dictionary entries is repeated in the schema (e.g., whether an attribute or class is optional or required, and the number of times each occurs). Detailed information about what values should go into the fields in the label is in general not in the schema, but may be found by following hyperlinks in the dictionary.

Part II. The First Steps

5.0 OUTLINE THE BUNDLE

We introduce the ‘Voyager 2 Jupiter Encounter Data’ archive that was originally produced by the PPI node and delivered to the PDS as a PDS3 dataset. This will provide the basis for all of the specific discussions which follow.

Our spacecraft is the ‘Voyager 2’ (VG2) spacecraft and the instrument is the ‘Plasma Science Experiment’ (PLS) a plasma instrument designed to detect plasma conditions throughout the Voyager trajectory.

	Name	*Abbreviation / Acronym*
Spacecraft	Voyager 2	VG2
Instrument	Plasma Science Experiment	PLS

5.1 Collections in the Bundle

Refer to the Standards Reference and confer with your PDS discipline node to determine all of the required and appropriate collections for your bundles.

Using the archive from our sample data, the instrument team plans to submit a single archive bundle to PDS, for the Venus encounter data. The bundle will consist of five collections.

Archive Bundle:
- Browse Collection
- Context Collection
- Data Collection
- Document Collection
- XML Schema Collection

5.2 Directory Organization

Our sample archive bundle is a fairly simple bundle that uses a fairly simple directory structure. Since there are only a small number of collections, we elect to have one directory in the bundle root for each collection.

The bundle root must contain at least one file, the XML label file for the bundle product, and may only contain one additional file – an optional readme file which if used is described in the bundle XML label file.

```
bundle root
| - bundle.xml
|
| - browse
|    | - collection_browse.xml
|    | - collection_browse_inventory.tab
|    | - collection product1
|
| - context
|    | - collection_context.xml
|    | - collection_context_inventory.tab
|
|    | - context_product1
|    | - context_product2
|    | - context_product3
|
| - data
|    | - collection_data.xml
|    | - collection_data_inventory.tab
|    |
|    | - data product1
|
| - document
|    | - collection_document.xml
|    | - collection_document_inventory.tab
|    |
|    | - doc_product1
|    | - doc_product2
|    | - doc_product3
|
| - schemas
|    | - collection_xml_schema.xml
|    | - collection_xml_schema_inventory.tab
|    |
|    | - xml_schema_product1
|    | - xml_schema_product2
|    | - xml_schema_product3
```

The root level subdirectories each correspond to a single collection. Each directory will contain the collection XML label file and the collection inventory file. Depending on the size of the collection, it may or may not contain other files.

In our example data, there is a single file in the data directory, ELEMON.TAB.

However, had there been multiple observational data files, it would have been reasonable for the team to decide to use the spacecraft clock count at the start of each observation as the primary reference for each observation. This would have been used as the filename root and in the logical Identifier (LID) for each observational data product. There are several approaches to setting names for each observational product and the corresponding subdirectories.

5.3 Determine the Documentation Needed

Refer to the Standards Reference and confer with your PDS discipline node to determine all of the required and appropriate documentation for your document collection(s).

You and the consulting PDS node should agree on a list of required documentation early in the design process. Documentation considered essential to understanding or using the archive and the underlying data in the archive, except for published journal articles, must be submitted as part of the archive. Journal articles may be included if permitted by the copyright holder. Each document to be archived must be prepared and saved in a PDS-compliant format, Refer to Section 11 of the Standards for a list of PDS approved formats.

In our archive example, documentation includes the following documents:

1. An errata file that describes any changes or errors in the archive.
2. A copy of a published journal article that describes the mission (in both ASCII and HTML).
3. A copy of a published journal article that describes the instrument (in both ASCII and HTML).
4. A checksum file that lists the MD5 checksum of the files in the archive. Note that this file is not a required PDS4 document. It is included in the PDS4 archive simply because it was part of the original PDS3 data_set.

Each of the above document products is individually labeled. Both the errata and the checksum files were each labeled using the Product_File_Text schema as these documents are strictly ASCII text. The other two document products, the mission and instrument descriptions, were each labeled using the Product_Document schema as these documents are presented in both an ASCII and an HTML version. Note that the two forms of the document, the ASCII and HTML versions, are collectively a single document product (i.e., All versions of a document are considered part of a single PDS document product).

6.0 DESIGN LOGICAL AND VERSION IDENTIFIERS

Every product label contains an identifier which must be unique across all products archived with the PDS. This identifier is referred to as a LIDVID and is the concatenation of a Logical Identifier (LID) and a version identifier (VID). We'll address the construction of each in the following sections.

6.1 General Concepts

Here are some general rules:

- LIDs must be unique across PDS
- Each PDS4 LID is constructed as aUniform Resource Name (URN)
- Each LID in a PDS archive begins with 'urn:nasa:pds'
- LIDs are case insensitive.
- LIDs are restricted to ASCII letters and numbers, dash, underscore, and period. Colons are also used but only in a very prescribed way discussed in this section.
- LID maximum length is 255 characters.

The complete set of requirements for LID construction is given in Section 7 of the PDS4 Standards Reference.

6.2 Constructing LIDs

Detailed requirements and formation rules are provided in the PDS Standards Reference, Section 7.1; we provide a brief summary here.

Recall that each basic product is delivered to PDS as a member of a collection, and that collection is a member of a bundle. LIDs are constructed based on a hierarchical set of relationships.

We can think of LIDs as constructed by concatenating fields of characters. The fields are separated by colons. This is the only use of colons permitted in LIDs.

- Bundle LIDs -- are constructed by appending a bundle specific field to 'urn:nasa:pds'.

  Bundle LID = urn:nasa:pds:<bundle field>

  Since all PDS bundle LIDs are constructed this way, the bundle field must be unique across all products archived with the PDS.

- Collection LIDs -- are constructed by appending a collection field to the parent bundle's LID.

  Collection LID = urn:nasa:pds:<bundle field>:<collection field>

  Since the collection LID is based on the bundle LID which is unique across PDS, the only additional condition is that the collection field must be unique across the bundle.

- Basic Product LIDs -- are constructed by appending a product field to the parent collection's LID.

  Product LID =
  urn:nasa:pds:<bundle field>:<collection field>:<product field>

  Since the product LID is based on the collection LID which is unique across PDS, the only additional condition is that the product field must be unique across the collection.

6.2.1 Examples

The following examples are based on a hypothetical mission.

	Name	*abbreviation*
spacecraft	Super SpaceCraft 01	ssc01
instrument	High Resolution Photon Counter	HiResPC
cruise phase	Cruise, Mercury, Earth phase	Cruise

The team decides to use the spacecraft clock count at the start of each observation as the product field of the LID for observational data products.

This is all the information we need to start designing LIDs.

Cruise Phase
Bundle
urn:nasa:pds:ssc01.HiResPC.Cruise

Collections
urn:nasa:pds:ssc01.HiResPC.Cruise:Browse
urn:nasa:pds:ssc01.HiResPC. Cruise:Context
urn:nasa:pds:ssc01.HiResPC. Cruise:Data
urn:nasa:pds:ssc01.HiResPC. Cruise:Document
urn:nasa:pds:ssc01.HiResPC. Cruise:Schema

Products [data products for sclock = 31234567]

urn:nasa:pds:ssc01.HiResPC. Cruise:Browse:browse_31234567
urn:nasa:pds:ssc01.HiResPC. Cruise:Data:data_raw_31234567
urn:nasa:pds:ssc01.HiResPC. Cruise:Data:data_derived_31234567
urn:nasa:pds:ssc01.HiResPC. Cruise:Document:errata
urn:nasa:pds:ssc01.HiResPC. Cruise:schema:Table_Character_0411f

6.3 VID Construction

Detailed requirements and formation rules are provided in the PDS Standards Reference, Section 7.2; we provide a brief summary here.

Version IDs are used for all types of products, including basic products, collections, and bundles.

- Version IDs must be of the form M.m where "M" and "m" are both integers. "M" is the "major" component of the version and "m" is the "minor" component of the version.
- The major number is initialized to one for archive products. (Zero may be used for sample products or test run products that are not intended for the archive.) The minor number is initialized to zero.

The VIDs in all of the products in our sample archive (e.g., bundle, collection, and product) is 1.0 which corresponds to the first submission for the archive (i.e., this version is the post peer-review, lien resolved version which is the first version that will be archive with the PDS and will go into the deep archive). Preliminary versions may not use version 1.0.

6.4 LIDVID Construction

Concatenate the LID and VID using a double colon as the connector. Here are sample LIDVIDs based the example LIDs in Section 6.2.1

6.4.1 Examples

Cruise Phase

Bundle

urn:nasa:pds:ssc01.HiResPC.Cruise::1.0

Collections

urn:nasa:pds:ssc01.HiResPC.Cruise:Browse::1.0
urn:nasa:pds:ssc01.HiResPC. Cruise:Context::1.0
urn:nasa:pds:ssc01.HiResPC. Cruise:Data::1.0

urn:nasa:pds:ssc01.HiResPC. Cruise:Document::1.0
urn:nasa:pds:ssc01.HiResPC. Cruise:Schema::1.0

Products [data products for sclock = 31234567]
urn:nasa:pds:ssc01.HiResPC. Cruise:Browse:browse_31234567::1.0
urn:nasa:pds:ssc01.HiResPC. Cruise:Data:data_raw_31234567::1.0
urn:nasa:pds:ssc01.HiResPC. Cruise:Data:data_derived_31234567::1.0
urn:nasa:pds:ssc01.HiResPC. Cruise:Document:errata::1.0
urn:nasa:pds:ssc01.HiResPC. Cruise:schema:Table_Character_0411f::1.0

6.5 LIDVIDs – The Next Step

LIDs and LIDVIDs will be ubiquitous in your archive. Be sure you have them correct before proceeding. When you have constructed draft LIDs and LIDVIDs contact your PDS DN to verify they are unique and conform to the requirements.

Once you and your DN have settled on a naming convention / formation rule for applying LIDs and LIDVIDs to the various products in your archive, the next step is to apply the formation rule to your pipeline that automatically generates the labels in your archive.

PART III. BASIC PRODUCTS – FROM SCHEMA TO LABELS

7.0 BASIC PRODUCT LABELS - AN OVERVIEW

The labels for basic products (all products except Collections and Bundles) structurally fall into two groups: document products and all other basic products. We start with all basic products other than document products; then we will discuss the aspects of document product labels which differ.

7.1 Selecting the Appropriate Generic Product Schema

The data provider and a representative from the DN discuss the anticipated nature of the data. Once the DN has sufficient information, they will select the appropriate generic schema, make modifications as appropriate to generate a tailored schema, and then they will pass the schemas to you, the data provider. Currently, for basic products there are 9 generic product schemas from which the DN will choose:

For binary array data:

- Product_Array_2D_Image
- Product_Array_3D_Image
- Product_Array_3D_Movie
- Product_Array_3D_Spectrum

For binary tabular data:

- Product_Table_Binary
- Product_Table_Binary_Grouped

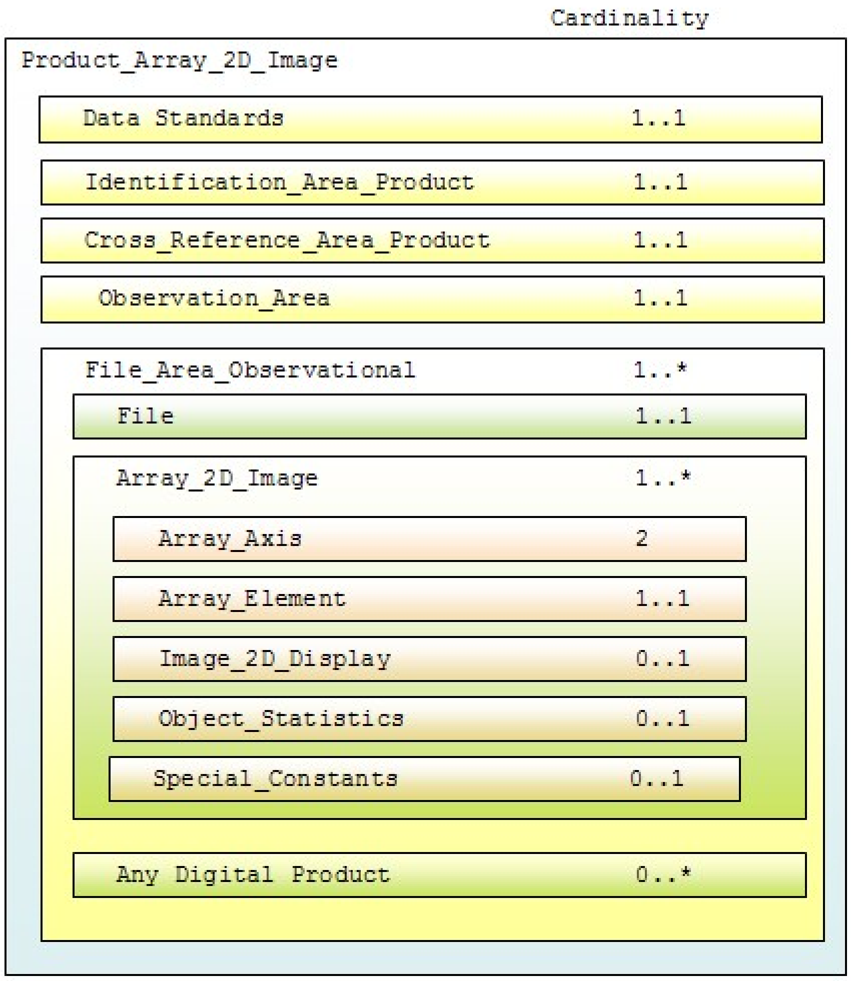
For character tabular data:

- Product_Table_Character
- Product_Table_Character_Grouped
- Product_Stream_Delimited

In addition there is a tenth generic product schema which does not identify a specific primary class such as Array_2D_Image or Table_Binary:

- Product_Non_Specific

Any basic product label may support multiple objects and multiple object types. While the first nine generic schemas listed above identify a specific object type as the primary product type for the product, all ten of the schemas provide hooks for all of the allowed object types.



The above diagram depicts a class hierarchy of the Product_Array_2D_Image schema. The hook to the other allowed object types is via the 'Any Digital Product' class within the File_Area_Observational area which allows the inclusion of any additional digital objects that may be secondary to the Array_2D_Image (e.g., Header, Table_Character, etc). For additional information, consult with your DN.

7.2 Basic Product Label Organization

All PDS XML labels begin with a small section of statements which can be considered XML overhead. In a typical basic product label these are followed by five major blocks of information (areas).

Each of these areas contains one or more classes each of which may have several attributes. The five areas are Data Standards, Identification Area, Cross Reference Area, Observation Area, and File Area.

- XML Overhead
- Data Standards
  - Gives the versions for the PDS Data Dictionary and Standards Reference to which the label conforms.
- Identification Area
  - Provides identification information for the product.
- Cross Reference Area
  - Provides identification information for products, journal articles, etc., relevant to understanding the product.
- Observation Area
  - Provides metadata classes describing the specific observation, laboratory experiment, etc.
  - Includes subsections for relevant classes specific to one or more discipline nodes, and to the mission (or equivalent name space).
- File Area
  - Identifies the file(s) containing the data object(s), and
  - Provides metadata classes specific to each data object in a given file (e.g., the description and parameters of each header, table, image).

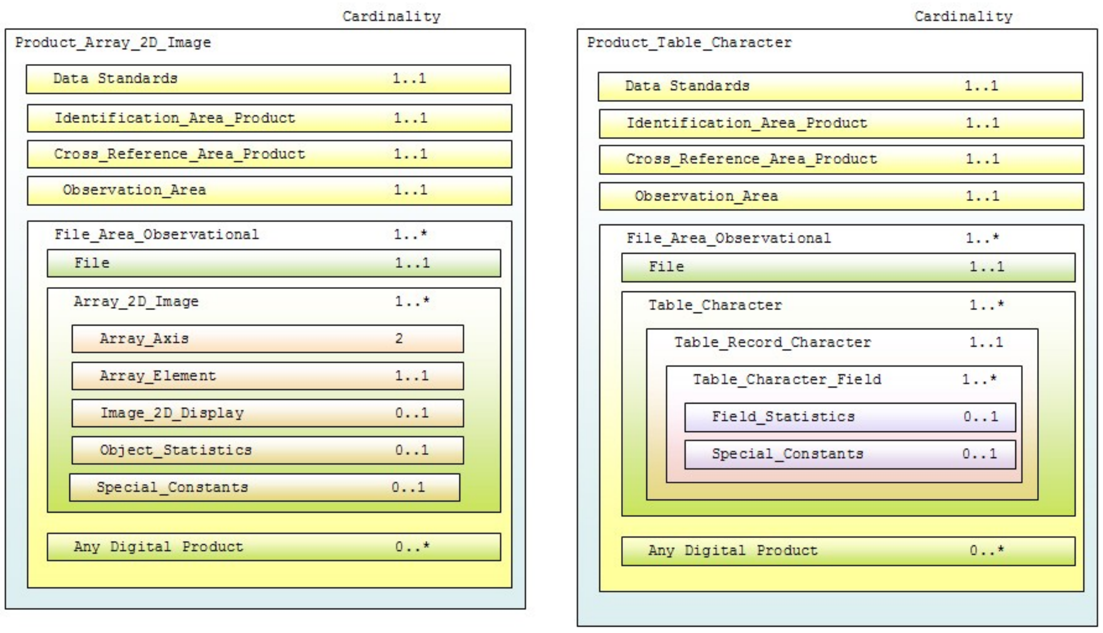


Figure 7-1. Basic Product (other than Document) Label Organization

Figure 7-1 depicts block diagrams for two different basic products. Notice that at the high level in this view the only differences relate to the first object described within the File Area. This structural consistency enables us to look at the construction of the label for a single basic product type in detail with the knowledge that the same techniques will be applicable across all label types. We will use Product_Table_Character for the following discussion.

Note that Figure 7-1 only reflects the PDS specific content of the labels with respect to the class hierarchy. There is a significant amount of content that is not depicted including the attributes contained in the classes and several lines of text preceding the information depicted here which are overhead associated with the use of XML.

Appendix B, Reading a PDS4 XML Schema, provides a basic overview of how to read and interpret a PDS4 schema.

8.0 SCHEMA EDITING

The majority of the schema editing probably will have been done by the DN in advance of you getting the schemas. The DN, in discussions with you, will have selected the generic schemas that correspond to the types of observational data and supplementary data that will be part of your archive submission to the PDS. The DN will most likely have tailored the generic schemas to more closely fit the particular nature of your supplementary data and in particular your observational data.

You should expect to receive the following from your DN:

1. A set of generic schemas --- one schema for each type of archival product
2. A set of tailored schemas --- one schema for each product where the generic schemas could be tailored to more closely fit the particular nature of your products. Note that the tailored schemas will probably be a subset of the generic schemas.
3. Three system dictionary schemas --- Attribute_Types, and Base_Types, and Class_Types. these are used in support of the generic and tailored schemas.
4. One or more discipline specific dictionary schemas --- possibly a mission-dictionary schema and / or a node-dictionary schema. These schema identify node and mission classes and / or attributes that are specific to the particular nature of the observational products that you are archiving.

8.1 Detailed Label Editing Information

For more detailed information on editing an XML schema, including a block by block description of editing each major section of the schema for an observational product, see Appendix C. Please note that Appendix C has not been updated, so there will be differences between the examples and the current generic schema.

9.0 LABEL TEMPLATE EDITING

The XML label template, as the name implies, is an xml file that will be used as a template for generating / producing the actual XML label(s) that will be used in your archive.

We can use the tools provided in an XML aware editor to generate a label template from our tailored schema, if there is a tailored version of the schema. Otherwise, we will generate a label template using the generic schema. You will generate a label template for each type of product that will be used in your archive (bundle, collection, and each variation of basic product).

The XML label template is an xml file which looks much more like the final label than does the schema. In the template, values which vary from one label to the next will be represented by placeholders that either you or the pipeline software will replace. In the examples in this and following Sections, we show fragments of XML templates. These generally contain dummy values inserted by the XML editor, for example name1 in
<name>name1</name>.

Values which are identical across all labels may be set to their appropriate value in the template.

In terms of making changes / editing the label template, there aren't that many types of modifications that can be made:

1. You can insert or remove classes or attributes designated as optional in the parent schema.
2. You can change the number of times a class or attribute is repeated
3. You can set the value of an attribute to be fixed / static
4. You can specify a value from an enumerated list of values

Keep in mind that all changes / modifications to the label template must adhere to the constraints dictated by the referenced schema.

We will address each of these types of modifications in the sections that follow.

9.1 Presence / Absence of a Class or Attribute

Recall that in a schema you indicate the presence and / or absence of a class or attribute by setting the minOccurs and maxOccurs values appropriately (i.e., Setting "minOccurs=0" and "maxOccurs=0", the XML element is not expected to be present in the label template).

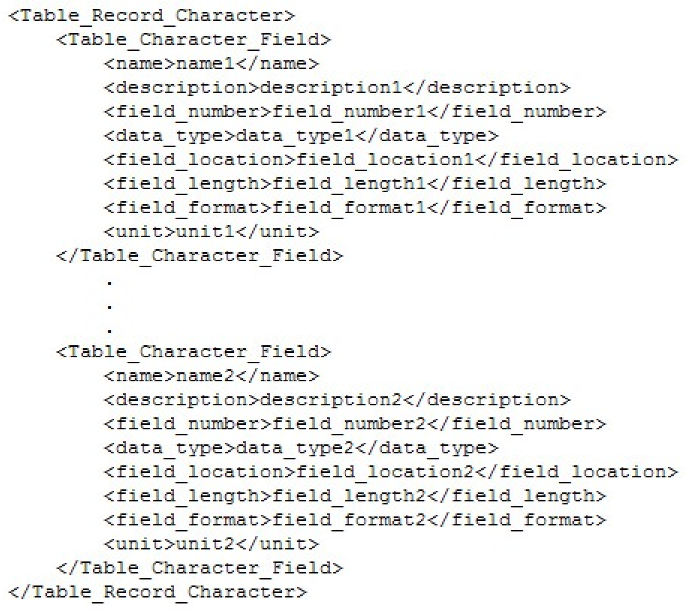
You have the same control over the presence and absence of classes and attributes within the label template. Use either your favorite text editor or XML aware editor to edit the label template and simply add or delete the class or attribute from the label template. Note that using an XML aware editor allows you to validate in real time as you make your edits.

9.2 Number of Times a Class or Attribute is Repeated

Recall that in a schema you may indicate the number of times a class or attribute will repeat by setting the minOccurs and maxOccurs values appropriately. For example, you can specify there are 5 , and always 5, fields in one row of a table by setting “minOccurs=5” and “maxOccurs=5”.

Because of the inherent nature of XML schema, you may not repeat a specific XML element. Contact your DN for more information on the limitations of replicating classes and attributes in XML schema.

However, this restriction does not apply to xml files and it is perfectly legal to repeat classes and attributes within the label template as long as we adhere to the restrictions specified in the reference schema.



```
<Table_Record_Character>
    <Table_Character_Field>
        <name>name1</name>
        <description>description1</description>
        <field_number>field_number1</field_number>
        <data_type>data_type1</data_type>
        <field_location>field_location1</field_location>
        <field_length>field_length1</field_length>
        <field_format>field_format1</field_format>
        <unit>unit1</unit>
    </Table_Character_Field>
        .
        .
        .
    <Table_Character_Field>
        <name>name2</name>
        <description>description2</description>
        <field_number>field_number2</field_number>
        <data_type>data_type2</data_type>
        <field_location>field_location2</field_location>
        <field_length>field_length2</field_length>
        <field_format>field_format2</field_format>
        <unit>unit2</unit>
    </Table_Character_Field>
</Table_Record_Character>
```

9.3 Set the Value of an Attribute to be Fixed / Static

Recall that in a schema you may set a fixed / static value for an attribute by inserting “fixed=<value>” in the XML element. The modified entry would look like this:

```
<xsd:element name="product_class" type="pds:product_class"
      fixed="Product_Table_Character" minOccurs="1" maxOccurs="1">
```

```
    </xsd:element>
```

You have the same control over setting a fixed / static value in the label template. Use either your favorite text editor or XML aware editor to edit the label template and simply enter the value between the XML tags.

```
<product_class>Product_Table_Character</product_class>
```

9.4 Specify a Value from an Enumerated List of Values

Specifying / restricting a value from an enumerated list of values is very similar to setting a value to be fixed / static. The difference being that the value must correspond to one of the values in the enumerated list.

Recall that in a schema you may set a fixed / static value for an attribute by inserting "fixed=<value>" in the XML element. The modified entry would look like this:

```
<xsd:element name="unit" type="pds:unit" fixed="byte" minOccurs="0"
    maxOccurs="1">
</xsd:element>
```

Keep in mind that by setting a fixed / static value in the schema, every instance of the attribute will have that fixed value. This may not be the best approach if the attribute is repeated. This would be the case for <unit> appearing in the <Table_Character_Field> class where there are N fields in the table and where the unit is not always 'Angstrom' in every field.

In this case, you actually have more control over setting a fixed / static value (from a list of enumerated values) in the label template. Each instance can be set independently. Use either your favorite text editor or XML aware editor to edit the label template and simply enter the value between the XML tags.

```
<unit>Angstrom</unit>
```

Note that in all cases, the value or values that are specified for that particular attribute must correspond to one of the values in the enumerated set.

9.5 Detailed Label Editing Information

For more detailed information on editing an XML label template, including a block by block description of editing each major section of an observational product, see Appendix D. Please note that Appendix D has not been updated, so there will be differences between some entries in the examples and entries based on the current generic schema.

9.6 Next Steps

In any case, you will want to have a fully functional PDS4 compliant label template in place before proceeding onto the next step of integrating the pieces into the pipeline production of the various products in your archive.

TBD

PART IV. COLLECTIONS, BUNDLES, DELIVERY PACKAGES

10.0 COLLECTIONS

The next higher level in the organizational hierarchy of an archive is the Collection — an inventory of member products and an accompanying label (including an identifier). The inventory and label are known as a Collection Product. Products of similar type and content are grouped into a collection. Observational data, for example, will be gathered into an observational data collection; documents into a document collection; and supplementary data into a supplementary collection.

In consultation with your DN, for each product type, determine how best to group the products, that share common characteristics, into appropriate collections.

To define a collection, the archivist creates a collection inventory, a specific type of character table, that lists all the product label files (XML) and their logical identifiers (LIDs) that are part of the collection. The archivist then creates a collection label that describes the collection inventory.

A collection label contains a logical identifier (LID) that uniquely identifies the collection and provides the links between products that share common characteristics.

The products listed in the collection inventory must be physically located either in the same directory as the collection label, or in one or more subdirectories to it. If there are more than @@@ products in the collection, they must be placed in subdirectories.

An example can be found at the following url:

http://pds.jpl.nasa.gov/repository/pds4/examples/

10.1 Members of a Collection

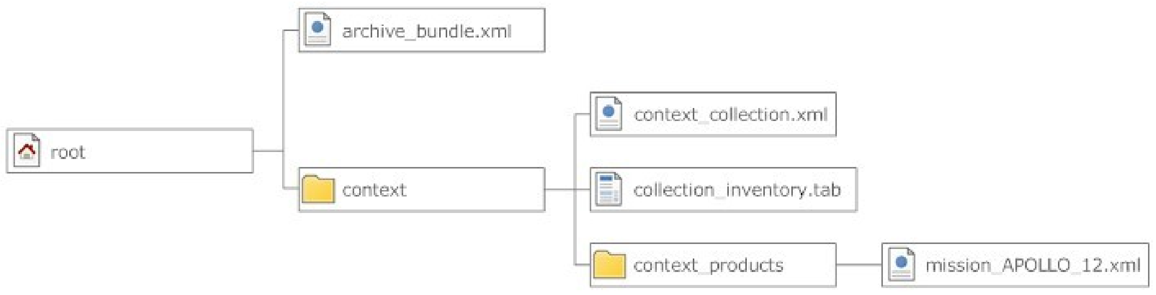
The Members of a collection are designated as being either primary or secondary.

- Every products must be a primary member of one and only one collection.
    - That collection is the one in which the product registered in the PDS repository.
- Products already registered in the PDS repository may be secondary members of other collections.

  - For example a collection of mosaic products might include the source products for each mosaic as secondary members of the collection.
- Primary members must be identified in the collection inventory using LIDVIDs.
- Secondary members may be identified in the collection inventory using either LIDs or LIDVIDs based on which is more appropriate for that collection and product.

10.2 Collection Inventory

Each collection product consists of two files, a collection XML label and a collection inventory. The collection inventory, a listing of the products that are the members of the collection, is a two column character table where each row of the table describes one of the member products of the collection. The first column of the table is the LIDVID of the product. The second column of the table is the file_specification_name of the product. The file_specification_name defines the name and location of the product relative to the location of the collection product.



In the above diagram for the mission_APOLLO_12 context product:

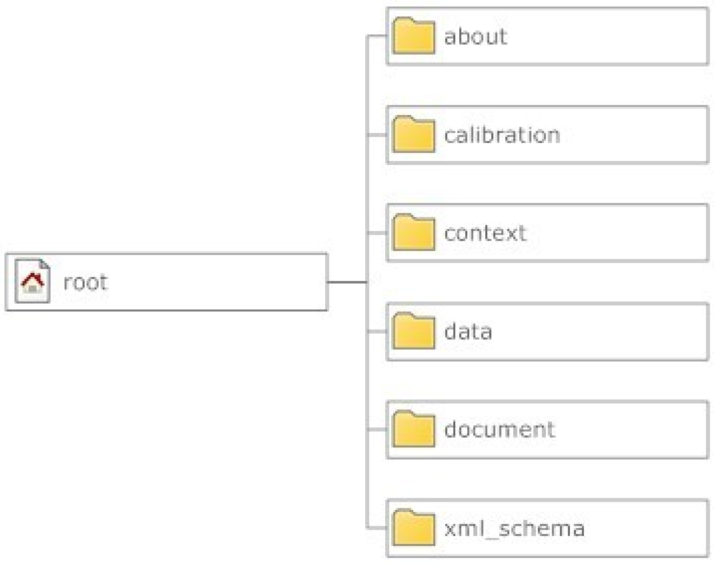
1. The LIDVID would be the value of the <logical_identifer> in the mission_APOLLO_12.xml file.
2. The file_specification_name would be “context_products/mission_APPOLO_12.xml”.

Note that directory paths are always specified in UNIX system format (i.e., the forward slash (“/”) is used to denote directories and to delineate subdirectories).

10.3 Generating and Populating a Collection Label

Using an XML aware editor, the archivist generates a label template using the Collection schema that most closely matches the type of products to be inventoried (e.g., you would use a Collection_Browse to inventory browse products; a Collection_XML_Schema to inventory XML schema products, etc).

You will create a collection product for each type of product in the archive. For example, if your archive has the following directory structure:



You would create a collection / inventory for the products contained in each of the above directories.

Directory Name	***Collection***
root	
About	Collection_Miscellaneous
calibration	Collection_calibration
context	Collection_context
data	Collection_data
document	Collection_document
xml_schema	Collection_xml_schema

10.4 Collection Product Label – Block by Block

10.4.1 Data Standards Area

The Data_Standards Area block in the template looks something like this:

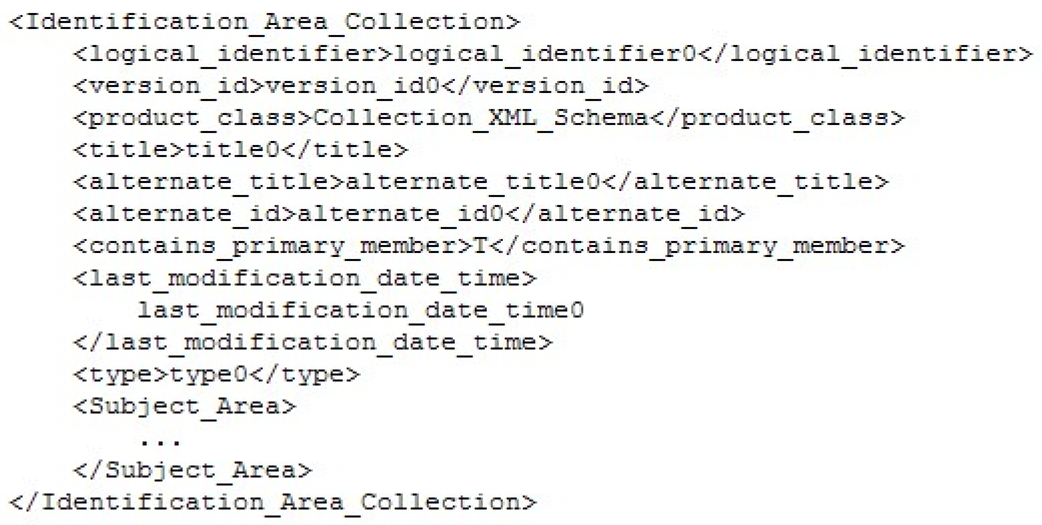
```
<Data_Standards>
    <dd_version_id>0.4.1.1.f</dd_version_id>
```

```
        <std_ref_version_id>1.0</std_ref_version_id>
    </Data_Standards>
```

The Data_Standards Area of the collection closely resembles that of the Basic Product.   See Section 4.n in the Standards Reference [2] for additional information on populating the Data_Standards Area.

10.4.2 Identification Area

The Identification Area block in the template looks something like this:



```
<Identification_Area_Collection>
    <logical_identifier>logical_identifier0</logical_identifier>
    <version_id>version_id0</version_id>
    <product_class>Collection_XML_Schema</product_class>
    <title>title0</title>
    <alternate_title>alternate_title0</alternate_title>
    <alternate_id>alternate_id0</alternate_id>
    <contains_primary_member>T</contains_primary_member>
    <last_modification_date_time>
        last_modification_date_time0
    </last_modification_date_time>
    <type>type0</type>
    <Subject_Area>
        ...
    </Subject_Area>
</Identification_Area_Collection>
```

The Identification Area of the collection closely resembles that of the Basic Product. The principle difference is that the Identification Area of the collection contains an additional attribute:

    <contains_primary_member>T</contains_primary_member>

The value is set to either 'T" or "F" depending upon whether or not the products being inventoried are primary or secondary members of the collection.   See Section 4.2 in the Standards Reference [2] for additional information on populating the Identification Area.

10.4.3 Cross Reference Area

Populating entries in the collection cross reference area is identical to that of the Basic Product. See Section 4.3 in the Standards Reference [2] for additional information on populating the Cross Reference Area.

10.4.4 Observation Area

Populating entries in the collection observation area is identical to that of the Basic Product. See Section 4.4 in the Standards Reference [2] for additional information on populating the Cross Reference Area.

10.4.5 File Area

The File Area of a collection product is used to reference primary and secondary products. If you are still uncertain as to whether or not the products to be inventoried are primary or secondary, consult your PDS discipline node.

We are going to use the example of a collection having both primary and secondary products. If your collection only has either primary or secondary products, then tailor the following steps accordingly.

There are two areas of the collection product XML label that we will use in order to inventory both primary and secondary products:

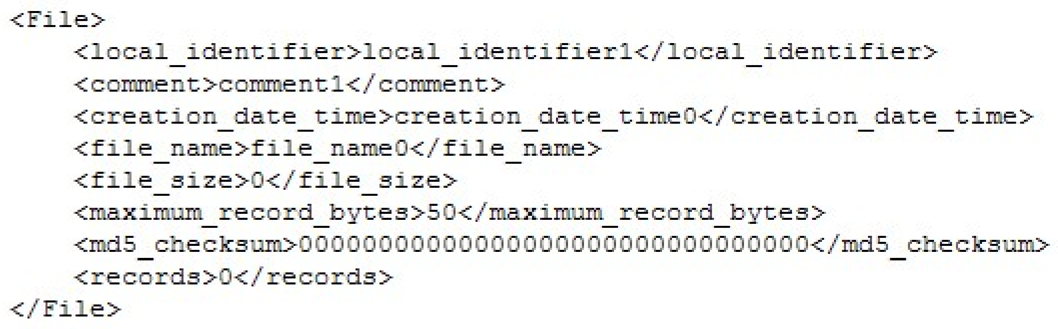
- File_Area_Inventory_LIDVID_Primary – used to inventory primary products
- File_Area_Inventory_LIDVID_Secondary – used to inventory secondary products.

In our example, your collection will consist of three files:
- collection.xml
- collection_inventory_primary.tab
- collection_inventory_secondary.tab

The collection.xml XML document will reference both the collection_inventory_primary.tab and the collection_inventory_secondary.tab files.
The collection_inventory_primary.tab will contain the inventory of the primary products and the collection_inventory_secondary.tab will contain the inventory of the secondary products.

10.4.5.1 File Area Inventory LIDVID (Primary and Secondary)

The two areas, File Area Inventory LIDVID (Primary and Secondary), are in essence identical and are populated identically. The only difference is that the two areas uniquely identify the type of product being inventoried --- primary or secondary.



```
<File>
    <local_identifier>local_identifier1</local_identifier>
    <comment>comment1</comment>
    <creation_date_time>creation_date_time0</creation_date_time>
    <file_name>file_name0</file_name>
    <file_size>0</file_size>
    <maximum_record_bytes>50</maximum_record_bytes>
    <md5_checksum>00000000000000000000000000000000</md5_checksum>
    <records>0</records>
</File>
```

Both areas contain a File area and an Inventory_LIDVID_* area. The File Area of the collection product XML label document is used to reference the data file(s) (i.e., the string of bits) being described by our collection label. In our example, there will be two File areas – one for each of the inventory files. The File area for a collection product is populated identically to that of a Basic Product. Refer to Section 4.5 in the Standards Reference [2] for additional information.

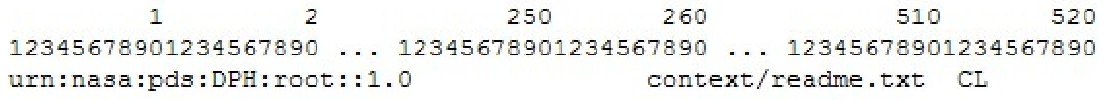
The Inventory_LIDVID (Primary and Secondary) areas provide the information that describes a simple two column character table where each row of the table describes one of the products in the collection. The first column of the table is the LIDVID of the product. The second column of the table is the file_specification_name of the product. The file_specification_name defines the name and location of the product relative to the location of the collection product.

The following label snippet is an example of how a LIDVID_Primary area is populated with metadata that describes a two column character table having 1 row of data, where each row is 512 bytes; each column contains 255 characters, and each row is terminated with <CR><LF>.

```
<File_Area_Inventory_LIDVID_Primary>
    <File>
        <local_identifier>inventory_context_collection</local_identifier>
        <creation_date_time>
           2011-08-23T11:53:24.5749Z
        </creation_date_time>
        <file_name>inventory_context_collection_20110823.tab</file_name>
        <file_size>2052</file_size>
        <md5_checksum>899fba057920491a08c7d517bd65e717</md5_checksum>
        <records>4</records>
    </File>
    <Inventory_LIDVID_Primary base_class="Table_Base">
        <local_identifier>dph_example_collection</local_identifier>
        <encoding_type>CHARACTER</encoding_type>
        <fields>2</fields>
        <offset>0</offset>
        <record_bytes>512</record_bytes>
        <records>13</records>
        <reference_association_type>
            has_member
        </reference_association_type>
```

```
            <Table_Record_Inventory_LIDVID_Primary>
            <Table_Field_LIDVID>
                <name>LIDVID</name>
                <description>
                    This columns specifies the LIDVID of the files that
                    comprise the collection.
                </description>
                <field_number>1</field_number>
                <data_type>ASCII_LIDVID</data_type>
                <field_location>1</field_location>
                <field_length>255</field_length>
                <field_format>urn:nasa:pds:xxxx::M.n</field_format>
            </Table_Field_LIDVID>
            <Table_Field_File_Specification_Name>
                <name>file_specification_name</name>
                <description>
                    This columns specifies the location of the files that
                    comprise the collection where the location is specified
                    relative to the location of the collection.
                </description>
                <field_number>2</field_number>
                <data_type>ASCII_File_Specification_Name</data_type>
                <field_location>257</field_location>
                <field_length>255</field_length>
            </Table_Field_File_Specification_Name>
            </Table_Record_Inventory_LIDVID_Primary>
        </Inventory_LIDVID_Primary>
    </File_Area_Inventory_LIDVID_Primary>
```

In the above label snippet, both columns have been set to 255 characters which is the maximum number of characters that a LIDVID and a file_specification_name can have. The two columns of data in the inventory file would have to be padded so that the 2nd column of data starts at byte position 257 and the <CR><LF> characters occupy positions 511 and 512..



```
          1         2             250       260             510       520
12345678901234567890 ... 12345678901234567890 ... 12345678901234567890
urn:nasa:pds:DPH:root::1.0              context/readme.txt   CL
```

An alternative, to padding the columns to the maximum length, would be to adjust the position of the 2nd column to start at a byte position that is greater than the actual maximum length of the values in the 1st column. For example, if the longest LIDVID is 50 characters in length, then the 2nd column could start at byte position 52.

10.5 XML Schema Collection

Probably one of the last collections that you will create will be the XML schema collection. This collection consists of an inventory of all of the schemas that were used in creating the products in your archive.

The collection label for the xml_schemas is virtually identical to the other collection labels.

10.6 Example Collection Product Label

11.0 SEE SECTION 16.BUNDLES

The highest level product is referred to as a Bundle — an inventory of member products and an accompanying label (including an identifier). Like collections, bundles consist of a list of references to products; however in this case, the referenced products are primarily collections. A bundle identifies all of the collections in the archive; where the collection; in turn, identifies all of the basic products necessary to perform useful science analysis on the data contained therein. The inventory and label are known as a Bundle Product. Unlike collection products, the bundle inventory is a table contained in the XML label file describing it, not in a separate table file.

[TBD Describe the differences / similarities between Product_Bundle and Product_Archive Bundle]

To define a bundle, the archivist creates a product XML label. A bundle label contains a logical identifier (LID) that uniquely identifies the bundle and provides the links between the collection products in the archive.

For as example see Section 16.

11.1 Generating and Populating an Archive_Bundle Label

Using an XML aware editor, the archivist generates a label template using the Product_Archive_Bundle schema.

You will create a single archive_bundle product for the archive with references to the member collections in the archive. Note that the archive_bundle product differs slightly from the bundle product in that the archive_bundle additionally specifies the information required for the 'readme.txt' file.

11.2 Data Standards Area

The Data_Standards Area block in the template looks something like this:

```
<Data_Standards>
    <dd_version_id>0.4.1.1.f</dd_version_id>
    <std_ref_version_id>1.0</std_ref_version_id>
</Data_Standards>
```

The Data_Standards Area of the bundle closely resembles that of the Basic Product. See Section 4.n in the Standards Reference [2] for additional information on populating the Data_Standards Area.

11.3 Identification Area

The Identification Area block in the template looks something like this:

```
  <Identification_Area_Bundle>
    <logical_identifier>logical_identifier0</logical_identifier>
    <version_id>version_id0</version_id>
    <product_class>Product_Bundle</product_class>
    <title>title0</title>
    <alternate_title>alternate_title0</alternate_title>
    <alternate_id>alternate_id0</alternate_id>

<last_modification_date_time>last_modification_date_time0</last_modification_date_time>
    <type>product_subclass0</type>
    <Subject_Area>
        …
    </Subject_Area>
  </Identification_Area_Bundle>
```

The Identification Area of the bundle closely resembles that of the Basic Product. See Section 4.2 in the Standards Reference [2] for additional information on populating the Identification Area.

11.4 Cross Reference Area

Populating entries in the bundle cross reference area is identical to that of the Basic Product. See Section 4.3 in the Standards Reference for additional information on populating the Cross Reference Area.

11.5 File_Area_Text

The File_Area_Text of an archive_bundle product XML label is used to reference the ‘readme.txt’ file (i.e., the string of bits) being described by the label.

.

The File_Area_Text class in the template looks something like this:

```
<File_Area_Text>
    <File>
        <local_identifier>local_identifier3</local_identifier>
        <comment>comment2</comment>
        <creation_date_time>creation_date_time0</creation_date_time>
        <file_name>file_name0</file_name>
        <file_size>0</file_size>
        <maximum_record_bytes>50</maximum_record_bytes>
        <md5_checksum>00000000000000000000000000000000</md5_checksum>
        <records>50</records>
    </File>
    <Stream_Text>
        .
        .
        .
    </Stream_Text>
</File_Area_Text>
```

Populating entries in the File_Area_Text of the XML label is remarkably straightforward. The file block consists of a number of optional and required attributes that describe the data file.

- The local identifier attribute, an optional attribute, provides the name of the local object being described; the value must be unique within the XML label.
- The comment attribute, an optional attribute, provides a brief description of the comment.
- The creation date time attribute, an optional attribute, provides the date/time when the file was created, either in YMD or DOY format.
- The file name attribute, a required attribute, provides the name of the file being described.
- The file size attribute, an optional attribute, provides the size (in bytes) of the file.
- The max record bytes attribute, an optional attribute, provides the size (in bytes) of the maximum record within the file. For example, in a character table having a fixed-length 78 byte record structure, the max_record_bytes is set to 78 (which is inclusive of the record delimiters).
- The md5 checksum attribute, an optional attribute, provides the md5 checksum of the file.
- The records attribute, am optional attribute, provides the number of records in the file.

The next block, the Stream_Text class, contains the reference to the ‘readme.txt’ product component being described by the XML label.

11.5.1 File Area - offset

Populating the offset attribute for the archive_bundle product is very simple. As there is only a single product being referenced, the offset is always set to '0' bytes.

```
<Stream_Text>
    <local_identifier>local_identifier1</local_identifier>
    <comment>comment1</comment>
    <encoding_type>CHARACTER</encoding_type>
    <external_standard_id>standard_id0</external_standard_id>
    <offset unit_of_measure_type="byte">0</offset>
</Stream_Text>
```

1. As the 'readme.txt' file is always ASCII, the encoding type is set to 'CHARACTER'.
2. The offset is set to indicate the starting location (in bytes) of the 'readme.txt' object, which in this case the header has a 0 byte offset.

11.5.2 File Area – data file location

The 'readme.txt' file referenced by the archive_bundle product label must be co-located with the XML label.

1. The file_name is simply set to the name of the 'readme.txt' file.
2. The file_size is set to the size of the 'readme.txt' file in bytes
3. The md5_checksum is set to the checksum of the 'readme.txt' file

Additional information on "The File Area" and how it is used within the archive can be found in the Standards Reference[2]. For specific uses of the File Area instances, you should consult the Data Dictionary[3a,b].

11.6 Archive_Bundle Area

Additional metadata is required to adequately describe the archive_bundle.

```
<Archive_Bundle>
    <start_date_time>start_date_time0</start_date_time>
    <stop_date_time>stop_date_time0</stop_date_time>
    <description>description1</description>
```

```
        <Citation>
            <citation_text>citation_text0</citation_text>
            <author_list>author_list0</author_list>
        </Citation>
    </Archive_Bundle>
```

1. The start_date_time attribute is set to the date and time at the beginning of a time interval of interest. Typically this is the date/time when the first observation was taken. The date/time format can be either YMD or DOY.
2. The stop_date_time attribute is set to the date and time at the end of a time interval of interest. Typically this is the date/time when the last observation was taken. The date/time format can be either YMD or DOY.
3. The description attribute is simply a terse description of the archive.
4. The citation class is repeated for each citation that is relevant to the archive.
5. The citation_text is simply a terse description of the citation.
6. The author_list attribute is set to a comma separated list of the name(s) of the composers of a work --- each author that contributed to the archive.

11.7 Bundle Member Entry

The Members of a bundle are referenced using the Bundle_Member_Entry area. The Bundle_Member_Entry area is repeated for each collection product.

```
<Bundle_Member_Entry>
  <file_specification_name>file_specification_name0</file_specification_name>
  <lid_reference>lid_reference1</lid_reference>
  <lidvid_reference>lidvid_reference1</lidvid_reference>
  <reference_association_type>has_member</reference_association_type>
</Bundle_Member_Entry>
```

In the Bundle_Member_Entry class:

1. You should use / specify either the lid_reference or the lidvid_reference attribute, but not specify both.
2. In all cases, the value for the reference_association_type is set to ‘has_member’.

The collection products referenced by a bundle label must either be co-located with the bundle label, or in a subdirectory of the directory containing the bundle label.

The Bundle_Member_Entry area is used to specify the location of the collection product. The value specified in the file_specification_name attribute is always relative to the location of the bundle label. If the bundle label and the collection label are co-located, then the value is simply the name of the XML label file:

```
<file_specification_name>filename.tab</file_specification_name>
```

If the collection XML label was located in the "test" subdirectory, then the value would be:

```
<file_specification_name>test/collection.xml</file_specification_name>
```

Note that directory paths are always specified in UNIX system format (i.e., the forward slash ("/") is used to denote directories and to delineate subdirectories).

12.0 DELIVERIES

Transfers of products from point to point within and external to the PDS can be accomplished using a variety of mechanisms from electronic transfer by email attachment of a handful of files to the transfer of 2TB of files using external hard drives. The actual transfer mechanism should be coordinated with the node with which you are working.

Any transfer, to, from, or within PDS has two parts, the "Package" containing the material being transferred and the "Manifest" describing those materials.

12.1 The Package

Transfer packages must be approved by PDS and include options such as ZIP, gzip, tar, physical media such as thumb drives, external hard drives, etc. There is no requirement to use any of these for the transfer of a handful of files, nor is there a requirement to use software such as zip when the entire transfer is being made with a dedicated external hard drive.

The data provider and the receiving node determine the best 'packaging' option for the delivery.

Except for the case of the delivery of a handful of files, either as test samples or as replacements for previously delivered files, the files within the delivery package must be organized in a PDS compliant directory structure (see the PDS Standards Reference, Section 3 for details.

12.2 The Manifest

The manifest is a file provided to PDS external to the package. It is a checksum manifest for each file in the package. Please note if the delivery package is a zip file, tar file, etc., the manifest must contain entries for every file in the package once that package is unpacked. Currently PDS uses MD5 checksums, so the delivery manifest is just an MD5 checksum manifest. Since it is external to the package, and hence external to the archive, there is no requirement to provide a PDS4 XML label for a delivery manifest.

12.3 Delivery of Accumulating Archives

The packaging and manifest constraints are exactly the same for incremental deliveries. However, it is neither required, nor in general desirable, for a provider to redeliver files which have not changed.

Consider a single collection with new files submitted quarterly to the PDS using an external hard drive. Each incremental delivery will have an updated collection product in the root directory, a sufficient portion of the collection directory structure for each of the files in the delivery to be placed appropriately, and all of the new files. It will also include a delivery manifest for that delivery; however the manifest must be provided to the PDS separately – not on the same external hard drive. [TBD – Is this correct? We definitely do not want the manifest inside the zip file, but can it be placed in the root directory of an external hard drive, or do we achieve improved data integrity by requiring it to remain separate?]

See the PDS Standards Reference, Section 7, for rules regarding Version ID incrementing for incremental deliveries.

13.0 LOCAL DATA DICTIONARY

A Data Dictionary serves several purposes. First, the dictionary serves as a reference manual to users of the PDS (and other planetary data systems) to define the attributes and classes that are used to describe planetary data and meta-data. Second, the dictionary serves as a reference for data producers (and others) to aid in the design and understanding of data descriptions. Third, the dictionary has the overall responsibility of ensuring the attributes and classes used in the data descriptions are used in a standard, consistent, predictable manner to the point where each attribute and class can be managed and used as a resource.

Conceptually, a data dictionary defines the attributes and classes which may be used in PDS4 product labels. Practically speaking, it must contain human-readable definitions as well as the syntax and semantic constraints placed on values of the attribute. For classes, it provides the explicit list of attributes constituting the class, and indicates which are required, optional, and/or repeatable. It might also indicate that one or more sub-classes are allowed (or required).

Every attribute and class that is used in any PDS label must first be defined in a data dictionary. Ultimately, all dictionaries will be integrated into the PDS4 Information Model which will "build" the PDS4 Data Dictionary document and the associated Mission and Node "dictionary" schemata.

Data dictionaries are categorized into:

- Global
- Mission specific
- Node specific

Global dictionaries are those that are used globally across all product schemata.
The global dictionaries are comprised of attributes and classes that are pervasive across a large number of the PDS product schemata.

Mission specific data dictionaries are those that are comprised of attributes and classes specific to a particular mission or investigation (i.e., the Mars Express mission could create a data dictionary for the sole purpose of defining attributes and classes that describe instrumentation and science data that is particular to the Mars Express mission). Depending upon preference, the mission may elect to have a single data dictionary (that describes both instrumentation and science data) or multiple dictionaries where one dictionary is specific to instrumentation descriptors and another is specific science data descriptors.

Node specific data dictionaries are those that are comprised of attributes and classes specific to a particular PDS node. For example, the Rings node has attributes / classes that are particular to a large number of Rings products. The Rings node may identify a number of instances where it would be convenient to group Rings descriptors into one or more data dictionaries.

13.1 Building and Using Local Data Dictionaries

Local data dictionaries are categorized into:

- Mission specific
- Node specific

This section describes the inter-relationships between the “generic” and “specific” data dictionary schemas, the data dictionary label(s), and the dictionary service that creates the local data dictionary schema.

The dictionary service optionally merges the attributes and classes defined in the local data dictionary with the PDS4 Information Model. This will ensure that your locally defined attributes and/or classes will be contained in the next build of the PDS4 Data Dictionary and the next build of the Generic Mission & Node schemata.

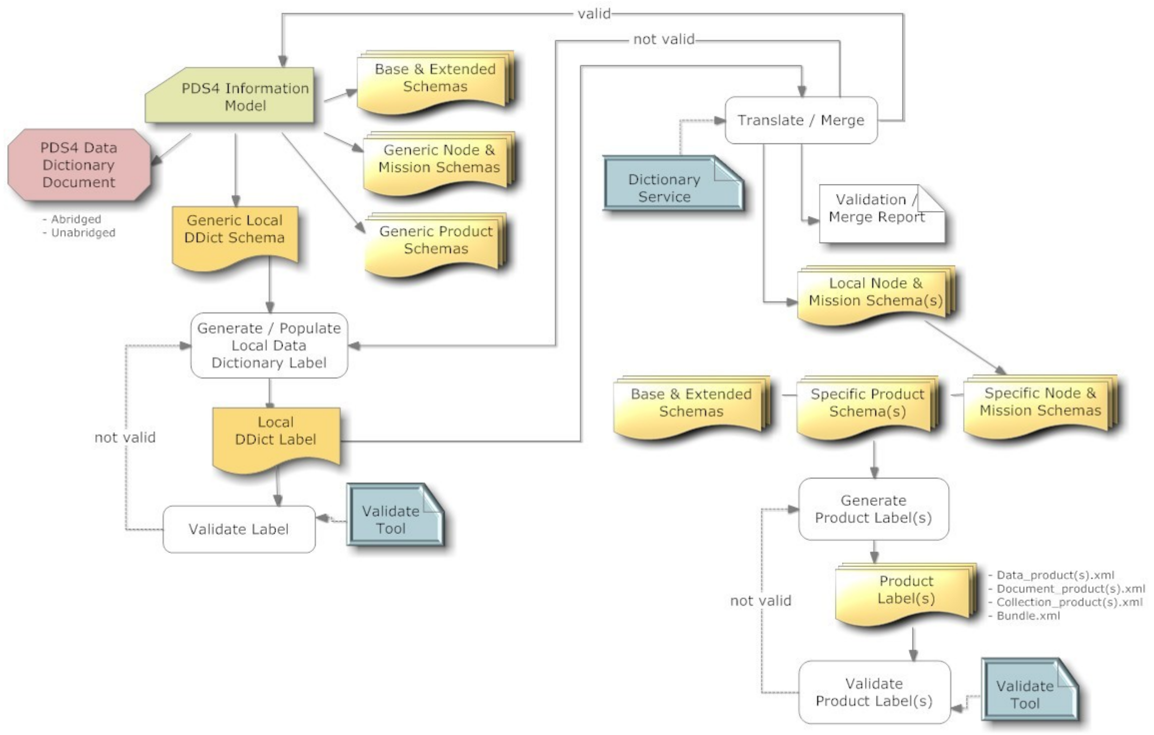

valid
not valid
PDS4 Information Model
Base & Extended Schemas
Generic Node & Mission Schemas
Generic Product Schemas
PDS4 Data Dictionary Document
- Abridged
- Unabridged
Generic Local DDict Schema
Generate / Populate Local Data Dictionary Label
Local DDict Label
not valid
Validate Label
Validate Tool
Translate / Merge
Dictionary Service
Validation / Merge Report
Local Node & Mission Schema(s)
Base & Extended Schemas
Specific Product Schema(s)
Specific Node & Mission Schemas
Generate Product Label(s)
Product Label(s)
- Data_product(s).xml
- Document_product(s).xml
- Collection_product(s).xml
- Bundle.xml
not valid
Validate Product Label(s)
Validate Tool

Figure 13-1 Local Data Dictionary Development Flow Diagram

Step #1: Select, from the set of PDS4 “Generic” Product Label Schemas, the Data-Dictionary schema, ‘Local_DD.xsd’.

Note that the Data-Dictionary schema contains an embedded reference to the first global dictionary, Extended_Types.xsd, which in turn has an embedded reference to the second global dictionary, Base_Types.xsd. You will need both dictionary files to be co-located to the Local_DD.xsd in order to proceed

Step #2: Download the Data-Dictionary schema, Local_DD.xsd, and the supporting schema Class_Types_nnn.xsd.

Step #3: Examine the as yet unmodified “Generic” Data-Dictionary schema in your favorite XML editor (e.g., Oxygen or Eclipse). You may also examine the schema in a text editor (e.g., UltraEdit, BBEdit, etc). Ensure that the XML is fully formed (i.e., the XML editor will validate the XML and will have an indicator (which is usually a green or red box) that indicates if errors are present in the XML).

Note that if there are errors in the XML schema, contact your PDS representative for further instructions on how to resolve any discrepancies.

In this case, the “Generic” schema represents the “Specific” overall structure and format of the data dictionary product. The “Generic” schema is immutable in that it defines, in the strictest sense, the greatest latitude permissible in the validation of the data dictionary product label to ensure PDS compliance (i.e., don’t even attempt to modify the “Generic” schema – everything is there that is needed to create new attributes / classes).

Step #4: Using your favorite XML editor (e.g., Oxygen or Eclipse), generate a local data dictionary XML label from the Data-Dictionary schema. Most XML editors provide a capability to “export/create” an XML label from an XSD. You will want to use this feature to export/create a sample label (which is an XML file) from the Data-Dictionary schema (which is an XSD file). Save the sample (as-yet-unedited) data dictionary label.

Step #5: Examine the as-yet-unedited data dictionary label in either your favorite XML editor or text editor. Ensure that the XML is fully formed (i.e., the XML editor will validate the XML and will have an indicator (which is usually a green or red box) that indicates if errors are present in the XML. As the sample label was generated by the XML editor, there shouldn’t be any errors. Contact your PDS rep to resolve any discrepancies.

Step #6: Using the XML or text editor, enter / populate the data dictionary label with the metadata associated with each attribute and class that is to be defined in the local data dictionary. Ensure that the XML is fully formed (i.e., the XML editor will validate the XML and will have an indicator (which is usually a green or red box) that indicates if errors are present in the XML).

You can use the PDS4 Validation Tool to further validate the contents of the XML label.

Step #7: At this point, you should have a fully formed compliant XML data dictionary label. A decision point has been reached whereby the contents of the locally defined data dictionary may be "merged" with the PDS4 Data Dictionary (via ingestion into the PDS4 Information Model).

To "merge" the locally defined data dictionary with the PDS4 Data Dictionary, set the "PDS4_merge_flag" to "true". If you choose not to "merge", then set the "PDS4_merge_flag" to "false".

Step #8: Access the online Data Dictionary Service, "point" the service to your data dictionary label, and press "go". The Data Dictionary Service will process your file and create a local instance of the "Node & Mission" schemata (XSD) and a report describing any anomalies encountered while translating / merging.

If you have elected to "merge" the locally defined data dictionary with the PDS4 Data Dictionary, the service will additionally validate the contents to ensure a compliant "merge" of the elements and classes is possible (e.g., no duplication of element / class names within the same Registration Authority, etc.). If the Service doesn't detect any anomalies, then the Service will "register" the contents of your data dictionary label with the PDS4 Information Model. This will ensure that your locally defined attributes and/or classes will be contained in the next build of the PDS4 Data Dictionary.

Step #9: Now that you have a "valid" local instance of the "Node & Mission" schemata (XSD), you can incorporate these dictionaries into your data product pipeline. This is done through an XML include reference in your specific product schema. So, the next step is to "link" the set of "Node & Mission" schemata (that are to be referenced) into the product schema.

Using the XML or text editor, add the equivalent of the following XML statements to the product schema.

```
  <xsd:include schemaLocation="Extended_Types_edited_0111c.xsd">
    <xsd:annotation>
      <xsd:documentation>PDS (common) Data Dictionary</xsd:documentation>
    </xsd:annotation>
  </xsd:include>

  <xsd:include schemaLocation="MarsExpress_instrumentation.xsd">
    <xsd:annotation>
      <xsd:documentation>
         Mars Express Data Dictionary for instrumentation
      </xsd:documentation>
    </xsd:annotation>
  </xsd:include>

 <xsd:include schemaLocation="Rings_Prod_Info.xsd">
    <xsd:annotation>
      <xsd:documentation>
         Rings Node Data Dictionary
      </xsd:documentation>
    </xsd:annotation>
  </xsd:include>
```

In the above, the first <xsd:include> instantiates a reference to the first global dictionary, Extended_Types.xsd. Recall, that Extended_Types.xsd has an embedded reference to the second global dictionary, Base_Types.xsd. The second <xsd:include> instantiates a reference to the "Mission" schema (that we created), "MarsExpress_instrumentation.xsd. The third <xsd:include> instantiates a reference to the "Node" schema (that we created), "Rings_Prod_Info.xsd.

Step #10: Now that you have all of the pieces in place, you can incorporate all of these files into your data product pipeline that will pump out gazillions of PDS compliant labels --- don't forget to validate, and then validate again, and again, and again…

Step #11: Repeat the above process for all the products in your archive. This includes most (but probably not all) of the following:

- bundle.xml
- collection_misc.xml
- aareadme.xml
- errata.xml
- collection_about.xml
- about_product(s).xml
- collection_browse.xml
- browse_product(s).xml
- collection_calibration.xml
- calibration_product(s).xml

- collection_context.xml
- context_product(s).xml
- collection_data.xml
- data_product(s).xml
- collection_document.xml
- document_product(s).xml
- collection_gazetter.xml
- gazetter_product(s).xml
- collection_geometry.xml
- geometry_product(s).xml
- collection_SPICE.xml
- spice_product(s).xml
- collection_xml_schema.xml
- xml_schema_product(s).xml

Step #12: The next step in the process, is to register all of the products in your archive.

[TBD See Section x.x.x for a description of the registration process.]

PART V. GENERATE THE ARCHIVE

14.0 LABELS, INVENTORIES, PACKAGES

[TBD]

15.0 VALIDATION

This section describes the process of validating the object-oriented design and the inherent relationships of and between the generic schema, the specific schema, and the resulting child XML document,

Figure 15-1 illustrates the process by which users ensure the resulting XML documents are compliant to the parent schemas. The validation process guarantees the object-oriented design of the parent-child relationships are preserved through out the design and implementation stages of preparing XML documents; specifically that:

1. The "Specific" Product schema validates/are valid against the "Generic" Product schema.

2. The "Label Template" validates/are valid against the "Specific" and the "Discipline Specific" schemas.

3. The PDS4 complaint labels validate/are valid against the "Specific" and the "Discipline Specific" schemas.

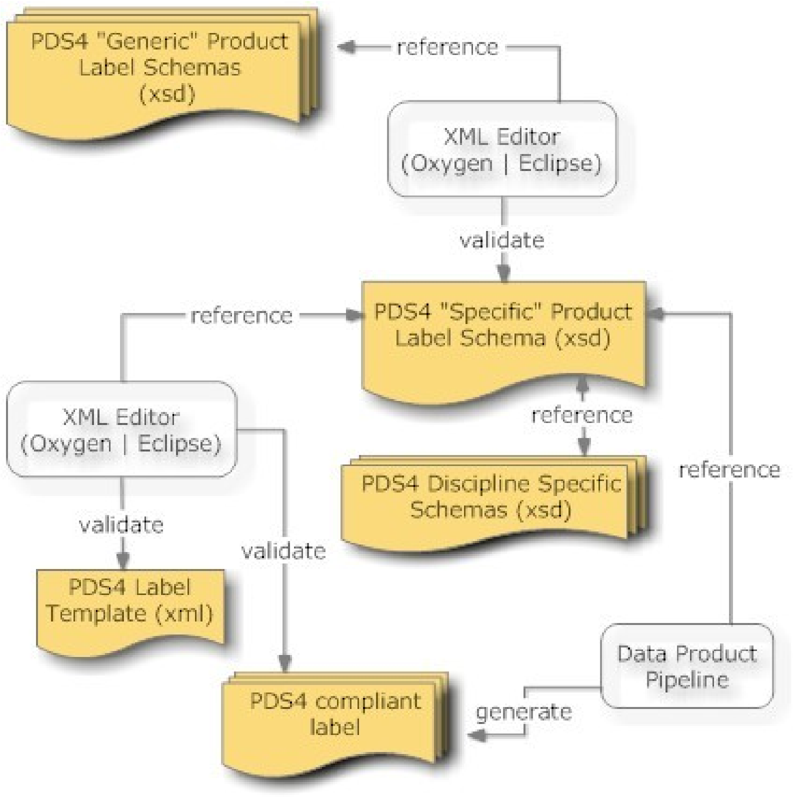

PDS4 "Generic" Product Label Schemas (xsd)
reference
XML Editor (Oxygen | Eclipse)
validate
reference
PDS4 "Specific" Product Label Schema (xsd)
XML Editor (Oxygen | Eclipse)
reference
PDS4 Discipline Specific Schemas (xsd)
reference
validate
validate
PDS4 Label Template (xml)
Data Product Pipeline
PDS4 compliant label
generate

Figure 15-1. Diagram of the Validation Lifecycle of a Product Label Schema

The underlying mechanism by which the above three steps is accomplished is noted as an XML editor (e.g., Oxygen or Eclipse). But, there are alternate mechanisms which could be used in place of your favorite XML editor; such as, an XML/XSD aware application. A machine-assisted mechanism for ensuring the “Specific” schema is valid against the “Generic” schema has yet to be determined.

PART VI. SAMPLES, EXAMPLES, OTHER TUTORIALS

16.0 EXAMPLE PDS4 PRODUCTS

A PDS4 tutorial would not be complete without providing a set of PDS4 products that were generated from example PDS3 products.

There are two different sets of examples:

- The first is a set of example products. This includes a representative set of products that would exist within an archive (e.g., character table, binary table, document, etc.).

- The second set is a complete archive. This includes all products that would comprise a complete PDS4 archive (e.g., archive_bundle, collections, collection inventories, readme, errata, and basic products). The archive is presented in a directory structure required of an archive.

  This material was originally archived under PDS and is representative of a PDS3 dataset migrated to become a PDS4 "example" archive. Consequently, the products within the "example" archive use "identifiers" that identify the products as "example.DPH" products (i.e., the "identifiers" are not representative of what an actual science archive would use).

  This is intentional and allows these products to be registered with the PDS registry service and searchable as "example products" and "example archive".

16.1 PDS4 Product Examples

@@@
The set of PDS4 product examples can be found at:

http://pds.jpl.nasa.gov/repository/pds4/examples/dph_examples_5g/

The HTML page that provides detailed descriptions of the PDS4 products can be found at:

http://pds.jpl.nasa.gov/repository/pds4/examples/dph_examples_5g/PDS4ExampleDataProductClasses.htm

Within this set of examples, there is an example of the following product types:

1. ARRAY_2D_IMAGE extension of the PDS4 Array_Base, (i.e., Homogeneous N-dimensional array of Scalars) class where a contiguous stream of BINARY data, assembled as a two dimensional data structure, maps the "items" contained in a ARRAY_2D_IMAGE file.

2. TABLE_CHARACTER extension of the PDS4 Table_Base (i.e., Heterogeneous repeating record of Scalars) class where a contiguous stream of ASCII characters, assembled as fixed-width fields, maps the "items" contained in a TABLE_CHARACTER file.

3. TABLE_BINARY extension of the PDS4 Table_Base (i.e., Heterogeneous repeating record of Scalars) class where a contiguous stream of BINARY data, assembled as fixed-width fields, maps the "items" contained in a TABLE_BINARY file.

4. TABLE_CHARACTER_GROUPED extension of the PDS4 Table_Base (i.e., Heterogeneous repeating record of Scalars) class where a contiguous stream of ASCII characters, assembled as sets of repeating fixed-width fields, maps the "items" contained in a TABLE_CHARACTER_GROUPED file.

5. STREAM_DELIMITED class where a contiguous stream of ASCII characters, combined with a field_delimiter and record_delimiter scheme, maps the "items" contained in a CSV "like" file.

6. DOCUMENT_SET class where one or more instantiations of a document (e.g., ascii text, pdf, html), as identified as a set, comprise a logically complete "copy" of the referenced document product.

16.2 PDS4 Example Archive

This set of PDS4 examples were derived from a PDS3 PPI data_set:

```
DATA_SET_ID = "VG2-J-PLS-5-SUMM-ELE-MOM-96.0SEC-V1.0"
```

Both the original PDS3 data_set files and the equivalent PDS4 product files are presented for the purposes of illustrating (comparatively) how a PDS3 data_set can be migrated to PDS4.

16.2.1 PDS3 Data_set Files

The directory structure of the original PDS3 PPI data_set consisted of the following:

- `aareadme.txt`
- `checksums.txt`
- `errata.txt`
- `\browse`
- `\catalog`
- `\data`
- `\documents`
- `\documents\mission`
- `\documents\pls`
- `\documents\symbols`

The files that comprise the original PDS3 PPI data_set can be found at:

- PDS3 PPI data_set files (directory listing):

http://pds.jpl.nasa.gov/repository/pds4/examples/dph_examples_5g /dph_example_PDS3_VG2PLS/

A zip file of the PDS3 VG2PLS data_set can be downloaded from:

http://pds.jpl.nasa.gov/repository/pds4/examples/dph_examples_5g /dph_example_PDS3_VG2PLS/VG2PLS.zip

16.2.2 PDS4 Product Files

The PDS4 archive structure, as migrated from the above PDS3 files, consists of the following:

- `Product_Archive_Bundle.xml`
- `Collection_sampleArchive.xml`
- `collection_inventory_20110823.tab`
- `README.TXT`
- `readme.xml`
- `\browse`
- `\context`
- `\data`
- `\document`
- `\schemas`

The following PDS3 files have been relocated to the “document” directory:

- `checksums.txt`
- `errata.txt`

The files that comprise the PDS4 migrated data_set can be found at:

- PDS4 migrated data_set files (directory listing):

http://pds.jpl.nasa.gov/repository/pds4/examples/dph_examples_5g /dph_example_archive_VG2PLS/

17.0 OTHER TUTORIAL MATERIAL

[TBD]

APPENDIX A ACRONYMS

The following acronyms are pertain to this document:

ADM	Architecture Development Method
API	Application Programming Interface
COTS	Commercial Off-The-Shelf
EN	Engineering Node (PDS)
ESDIS	Earth Science Data and Information System
FTP	File Transfer Protocol
IEEE	Institute of Electrical and Electronics Engineers
IPDA	International Planetary Data Alliance
IT	Information Technology
JPL	Jet Propulsion Laboratory
NASA	National Aeronautics and Space Administration
NSSDC	National Space Science Data Center
PDS	Planetary Data System
RM-ODP	Reference Model of Open Distributed Processing
RSS	Really Simple Syndication
SDSC	San Diego Supercomputing Center
SOA	Service-Oriented Architecture
TB	Terabyte
TOGAF	The Open Group Architecture Framework
XML	eXtensible Markup Language

APPENDIX B READING A PDS4 XML SCHEMA

In your XML-aware editor, or in your favorite text editor open Product_Table_Character_*.xsd from the example set.

The top of the schema is XML overhead and may require some editing.

[TBD] – provide specifics – when, why, how

The actual PDS4 label contents begin with the first <xsd:complexType …> statement. In the case of the schema for Product_Table_Character, this opening line is:

```
<xsd:complexType name="Product_Table_Character_Type">
```

This line is the beginning of an XML element. The complete XML element is everything from the opening line above to the corresponding closing XML tag, </xsd:complexType>.

Between the opening and closing XML tags, are four XML elements which identify the four areas used to describe an individual character table product:

```
<xsd:element name="Identification_Area_Product"  … </xsd:element>
<xsd:element name="Cross_Reference_Area_Product" … </xsd:element>
<xsd:element name="Observation_Area" … </xsd:element>
<xsd:element name="File_Area_Observational" … </xsd:element>
```

Each of these XML elements has four pieces of information: name, type, minOccurs, and maxOccurs.

- "Name" tells us what the Area is. For example the name="Identification_Area_Product" refers to the PDS4 class describing the identification area for all basic products.

  Case matters here. Camel case, which all of these use, indicates the entry is a PDS class which in turn will contain multiple attributes. If the value for name is all lower case the entry is a single PDS attribute.

- "Type" ultimately describes the contents of the class or attribute named. In the case of Identification_Area_Product, the value for type in the schema is:
  pds:Identification_Area_Product_Type.
  - Here the leading "pds:" indicates that what follows the colon belongs to the "pds" namespace.
  - Identification_Area_Product_Type is expanded in the schema in the section following the Product_Table_Character_Type section.

- "minOccurs" gives the minimum number of times the area described by the XML element can appear in a label.
    - For minOccurs, a value of 0 indicates the attribute described by the XML element is optional; a value of 1 or greater indicates the attribute is required.
- "maxOccurs" gives the maximum number of times the XML element can appear.

For the first three of these XML elements, both minOccurs and Max Occurs are preset to 1. The interpretation is that each of these areas must appear once, and only once, in a single label.

For File_Area_Observational, minOccurs is set to 1 (the XML element is required), and maxOccurs is set to "unbounded" (the XML element can occur an unlimited number of times). Hence the objects described by this label must be in at least one file, but may span several files.

Careful, while a product may span several files, and a single file may contain several objects (e.g., a header, an image and several character tables), each object must be contained within a single file (i.e., you cannot put the first half of a table in one file and the second half in a different file).

APPENDIX C EDITING AN XML SCHEMA

This appendix includes a block by block description of editing each major section of an XML schema for an observational product and how each section of the schema could be edited to tailor the schema to be more specific with respect to the nature of your particular archive.

The following sections describe a step-by-step introduction as to a possible methodology for approaching how to edit a schema. This is simply an introduction. Additional information / introspectives on editing schemas should be actively sought out from your DN.

NOTE: The example XML fragments have not been updated to the most recent generic schema, so some entries may not match exactly the current schema; however the process and considerations offered should still be valid.

Start a working directory, the minimum contents of which should be:

- The discipline node's tailored version of the schema you will be editing.
- A copy of the above, appropriately renamed (perhaps insert spacecraft and instrument IDs).
    - Open this in your XML aware editor.
- Schema for Attribute_Types, and Base_Types, and Class_Types
- Other included schema, such as a node or mission data dictionary

You will find it helpful to have the data dictionary open. Search the dictionary for Product_AAA (where AAA is the name of the primary class – e.g., Array_Image_2d, Table_Character). Select the entry with the full description of Product_AAA and scroll to the specific area portion of that entry for the area in which you are working (e.g., Identification Area).

- Make a copy of the Product_Table_Character schema, giving it a slightly different name than the example specific schema.
- Make edits as they are described in this document and compare to the example specific schema.

Recall that XML elements identified as optional in the tailored schema may be considered to fall into one of three categories, 'will be used', 'will not be used', or 'may be used'.
For the worked example, we handle optional XML elements as follows.

- For 'will be used'
    - If the number of times the XML element will occur is fixed, set both minOccurs and maxOccurs to that value.
    - If the number of times the XML element will occur is not fixed, set minOccurs to the highest nonzero value that represents the smallest number of occurrences, and maxOccurs to the lowest reasonable upper bound.
- For 'will not be used' delete the optional XML element.

- For 'may be used'
  - leave minOccurs set to zero
  - set maxOccurs to the lowest the lowest reasonable upper bound..

When you first open the schema you will be editing in an XML editor, if the editor can find all of the appropriate supporting files it will indicate that the current document is 'valid'. As you make edits, the editor will help you identify invalid entries and assist in repairing them. Schema editing can be done in a text editor such as BBEdit or UltraEdit, but such editors do not provide validation checking. For more information on XML editing and editors, see Appendix B.

C.1 Xml Schema Overhead

```
<?xml version="1.0" encoding="UTF-8"?>
  <!-- PDS4 XML/Schema for Product_Table_Character_0.4.1.1.f  Sat Aug 20 07:14:14 PDT
2011 -->
  <!-- Generated from the PDS4 Information Model V0.4.1.1.f -->
  <!-- *** This PDS4 product schema is a preliminary deliverable. *** -->
  <!-- *** It is being made available for review and testing. *** -->
  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://pds.nasa.gov/schema/pds4/pds/v04"
    xmlns:pds="http://pds.nasa.gov/schema/pds4/pds/v04"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified"
    version="0.4.1.1">

    <xsd:include schemaLocation="Class_Types_0411f.xsd">
      <xsd:annotation>
        <xsd:documentation>PDS (common) Data Dictionary - Tagged Data
Objects</xsd:documentation>
      </xsd:annotation>
    </xsd:include>

  <!-- <xsd:import namespace="http://pds.nasa.gov/schema/pds4/anyNS"
    schemaLocation="Any_Described_Data_Object_0411f.xsd"/> -->
```

[TBD Need to explain these and discuss what if need to add anything else and how to set up editor catalog]

C.2 Data Standards

This section contains two attributes: dd_version_id and std_ref_version_id.

```
<Data_Standards>
    <dd_version_id>0.4.1.1.f</dd_version_id>
    <std_ref_version_id>1.0</std_ref_version_id>
```

```
        </Data_Standards>
```

The dd_version_id attribute is set to the version identifier for the data dictionary that was currently in use when the archive was produced.

The std_ref_version_id is set to the version of the Standards Reference that was currently in use when the archive was produced.

C.3 Identification Area Product

This section of the schema has several XML elements and each applies directly to the product (e.g., version_id is the version ID of the product described in the label file). Let's go line-by-line.

- logical_identifier – required. Occurs once; the value is unique for each product, so we leave it alone at this stage.

- version_id – required. Occurs once. For a collection which is only going to be generated once, you may wish to set the value by inserting fixed= "1.0" in the XML element. The modified entry would be:

  ```
  <xsd:element name="version_id" type="pds:version_id_Type"
                    fixed="1.0" minOccurs="1" maxOccurs="1"> </xsd:element>
  ```

  However, if the collection is likely to be revised, or if it will be accumulating over time, it is probably best to leave the XML element unchanged in the schema and adjust the value in the data pipeline. We chose this latter option in the example file.

- product_class – required. Occurs once; the value is already fixed, so leave it alone.

- title – required. Occurs once. This is not a term that is fully self-explanatory, so we looked up the definition in the dictionary. From the dictionary, in this context, title is a name for the product (e.g., Cassini CIRS thermal scan, PHOENIX Mars Wind Experiment).

  As the title attribute will be used as the basis for searching for all products generated with the schema we are producing, we will need to create a formation rule whereby each product will be uniquely named. You may want to confir with your DN as to a suitable naming convention.

- alternate_title – optional. A place for us to provide additional names for the product; may be used more than once. If you think there will be different terms by which people will refer to / search for the products you are labeling with this schema, one of those is the title, and the other is the alternate title attribute. You may change both minOccurs and maxOccurs for alternate_title to “2”. When you change maxOccurs to “2”, you can set the options so that the editor will put two alternate_title entries in the XML template. Unfortunately the rules of XML will not let you have two alternate_titles in the schema, so you cannot use the “fixed=” approach where you need more than one of a particular XML element.

In our example, we don’t plan to use alternate_title, so we deleted the entry.

- alternate_id – optional. -Used in data sets which were archived under PDS3 and subsequently are being migrated to comply with PDS4. In those cases this is used to capture PDS3 specific identifiers such as a the PDS3 product_ID. The example product we are constructing is for data not previously archived under PDS3, so we deleted the entry.

- last_modification_date_time – optional. Although this attribute is optional in the generic schema; your PDS DN will probably have set it to required in the tailored schema they provided you.

- type – optional. Currently not used by the system; deleted in the example.

- Subject_Area – optional. Occurs once, so we leave this entry as is.

  Subject_Area is a PDS class, embedded within the Identification_Area. Subject_Area_Type is the block in the schema which expands this class and we may wish to make edits there.

  In the generic schema none of the XML elements contained in the subject area are required. This is because PDS acknowledges that no single XML element in this area will be appropriate for all possible basic products. The data provider, working with discipline node, will need to decide which are appropriate for this particular set of products.

  One function of the subject area is to make human readable identification information available in the label. Another function served by this area is to facilitate search and retrieval. Consequently for an observational data product, it is probably never appropriate to remove all of the contents. The underlying guideline is to do something reasonable.

  Again the discipline node will probably set some of the XML elements to required in the tailored schema they provided you.

  We’ll work through the section line by line.

- target_name – optional. In most cases will be changed to required either by the DN or the Data Provider. Since this entry will be used to facilitate search a reasonable amount of specificity is appropriate here. Notice that maxOccurs is unbounded – you are not limited to specifying only one target for the product.

  For the Imaging Science Subsystem on Cassini, setting target to "Saturn System" for all observations would not qualify as something reasonable. Each observation has an intended target or set of targets. We want to be as accurate and as comprehensive as possible in listing targets of observational data.

- data_set_name – optional. This would be used to reference / cite one or more PDS3 data_sets.

  We removed it in our example.

- instrument_name – optional. Generally, the name of the instrument used in the observational data.

- instrument_host_name – optional. Generally the name of the spacecraft, or for Earth based observations, the name of the telescope.

  While the value(s) for target will probably be populated by the pipeline, instrument name and instrument host names are good candidates for populating in the edited schema using "fixed=". We chose to do this in our example.

- full_name – optional. The dictionary indicates this is the full name of a person, generally the data preparer's name goes here, The underlying guideline is to provide a reference to the individual mainly responsible for the production of the archive.

- investigation_name – optional. If there is an appropriate investigation name, it goes here.

- observing_system_name – optional. May be used in lieu of the separate entries for instrument_name and instrument_host_name.

- Name_Resolution – optional. This is a placeholder for a concept that has not yet fully matured. Eventually this will be defined and appropriate for use. We removed this line from the examples, but that is only temporary.

C.4 Cross Reference Area Product

This section has three XML elements and each supports a PDS class; consequently each of those classes is expanded in the three blocks of XML immediately following this section. The three PDS classes are given in alphabetical order in the schema. However if they were ordered functionally, we would probably list the three as Observing_System, Bibliographic_Reference, and Product_Reference since Bibliographic_Reference and Product_Reference are complimentary.

Observing System is used to identify combinations of one or more subsystems used to obtain the data in the product (e.g., observatory + telescope + detector, spacecraft + instrument, laboratory facility + instrument).

Bibliographic References provide references to documents, relevant to the product, which are external to the PDS (e.g., a journal article describing the detailed processing used to produce the product).

Product References provide references to products which are in the PDS and relevant to this product (e.g., browse products, source products, calibration documents).

While observing system is required for most basic products, both bibliographic and product references are optional. These two reference classes are optional because there will be many instances when one or the other will appropriate, but not both. On the other hand, it is difficult to imagine a PDS product produced in such isolation that neither bibliographic nor product references are appropriate. Consequently you will want to consider changing minOccurs to "1" for one or both of these classes. In our example we changed the value for Product reference.

C.4.1 Bibliographic Reference

The Bibliographic Reference area is optional. This class is repeated for each document external to PDS to which you need to refer. For each such document, there are five attributes that can be used to identify the document:

- local_identifier – required. Only used once for each document you are referencing. Any "local" identifier is intended for use within the context of the label we are writing; it must be unique within the label. Here is the description from the data dictionary:

  The local identifier attribute provides the name of a local object; it is unique within a label. The value of local_identifier should be the class name. If several instances of the same class exist, then a numeric suffix is appended to the class name.

  In this case the class is "Bibliographic_Reference", so the first document will be Bibliographic_Reference1, the second will be Bibliographic_Reference2, etc.

- description – optional. Description of the subject matter of the document.
- doi – optional. If the cited document has a Digital Object Identifier give it here.
- reference_text – required. This is a complete bibliographic citation for the published work.
- url – optional. The URL to the cited document.

Here is an example of a bibliographic reference:

```
<Bibliographic_Reference>
    <local_identifier>reference.JPLPD618-128</local_identifier>
    <description>reference document</description>
    <doi>unk</doi>
    <reference_text>
           Voyager Neptune/Interstellar Mission: Flight Science Office Science and Mission
           Systems Handbook, JPL Project Document 618-128, Jet Propulsion Laboratory,
           Pasadena, CA, 1987.
    </reference_text>
    <url>unk</url>
</Bibliographic_Reference>
<Bibliographic_Reference>
    <local_identifier>reference.KOHLHASE1989</local_identifier>
    <description>reference document</description>
    <doi>unk</doi>
    <reference_text>
           Kohlhase, C., The Voyager Neptune Travel Guide, JPL Publication 89-24, 276 pp.,
           Jet Propulsion Laboratory, Pasadena, CA, 1989.
    </reference_text>
    <url>unk</url>
</Bibliographic_Reference>
<Bibliographic_Reference>
    <local_identifier>reference.MORRISON1982</local_identifier>
    <description>reference document</description>
    <doi>unk</doi>
    <reference_text>
           Morrison, D., Voyages to Saturn, NASA SP-451, 227 pp., National Aeronautics and
           Space Administration, Washington, DC, 1982.
    </reference_text>
    <url>unk</url>
</Bibliographic_Reference>
```

From the above, you will notice that the local_identifier is unique across the Bibliographic references. The local_identifier must be unique for all products in a label.

C.4.2 Observing System

The Observing System area is required. The Observing System area may be repeated if more than one observing system was used to collect the data. The use of multiple observing systems will probably be rare.

- Observing System – required.
  - local_identifier – optional. Works just the same as in bibliographic reference.
  - title – required. The name of the observing system.
  - description – optional. A brief text entry describing the observing system.
  - Observing_System_Component – required. Occurs may occur several times.

    The dictionary entry indicates that there are two types for the components, either sensor or source.

    - observing_system_component_type – required. Has two possible values: sensor or source.
    - Observing_System_Reference_Entry – required. Provides PDS identifiers for the observing system components, and any other PDS products associated with the observing system (e.g., descriptive documents).
      - lid_reference, lidvid_reference – required. The dictionary and schema provide separate entries for LID and LIDVID and each is listed as optional; however one of the two must be populated whichever makes the most sense for the particular observing system.
      - reference_association_type – required. This tells the PDS system what the relationship is between the observing system and the product for which you just provided a LID or LIDVID. The value is in the form “has_*”. From the dictionary entry we see that * can be one of three things: instrument, instrument_host, or association.

C.4.3 Product Reference Entry

The Product_Reference_Entry is optional. This class is the counterpart to the Bibliographic Reference Area for those references which are registered in PDS. This class is repeated for each PDS registered document to which you need to refer.

This is another variation of reference entry, so it is very similar to the Observing_System_Reference_Entry in the previous section.

- ▫ lid_reference, lidvid_reference – required. The dictionary and schema provide separate entries for LID and LIDVID and each is listed as optional; however, one of the two must be populated.

- ▫ reference_association_type – required. This tells the PDS system what the relationship is between the product described by the label generated from this schema and the product for which you just provided a LID or LIDVID. Most of the possible values are in the form "has_*". There are more than a dozen associations listed in the dictionary. Be sure that you find the variation of Product_Reference_Entry for Product_Table_Character to ensure you have the correct list of values.

C.5 Observation Area

The Observation_Area is required. This is the area where we describe the observation or experiment. If there is metadata (information about data) that is important to understanding or interpreting the data, it needs to accompany the data and most of it will be found in this section of the label.

There are several core PDS attributes in this section which are perceived as reasonably common and important. Clearly PDS can not identify a comprehensive set of attributes to provide all of the metadata necessary for every possible data file in PDS. Consequently, the Observation_Area includes two optional subclasses, a Node_Area and a Mission_Area which allow PDS discipline nodes and data providers respectively to expand the contents of the Observation_Area.

Now let's go through Observation_Area line-by-line. We will not expand on items which we have encountered before are for which the function is obvious.

- o comment – optional.

- o start_date_time – required. The date and time corresponding to the beginning of the observation. It is left as an exercise for the reader to determine the format options. If you run into problems, review the discussion of last_modification_date_time.

- o stop_date_time – required. The date and time corresponding to the end of the observation.

- o local_mean_solar_time – optional.

- o local_true_solar_time – optional.

- o mission_phase_name – optional.

- orbit_number – optional.
- planet_day_number – optional.
- solar_longitude – optional.
- spacecraft_clock_cnt_partition – optional.
- spacecraft_clock_start_count – optional.
- spacecraft_clock_stop_count – optional.

C.5.1 Mission Area

- Mission Area

  The Mission Area of the product label is a container for all mission-specific classes and attributes. These classes and attributes must be defined in a local data dictionary (see section 4.9) and must be prefixed with the namespace identifier applicable to that local dictionary.

```
<Mission_Area>
  <mpf:application_packet_id>34</mpf:application_packet_id>
  <mpf:application_packet_name>SCI_IMG_3</mpf:application_packet_name>
  <mpf:auto_exposure_data_cut>3000</mpf:auto_exposure_data_cut>
  <mpf:auto_exposure_pixel_fraction>1.0000</mpf:auto_exposure_pixel_fraction>
  <mpf:frame_id>BOTH</mpf:frame_id>
</Mission_Area>
```

The current thinking is that individual observational products will be specific to a single mission and so there shouldn't be a reason to have multiple Mission Areas in a product label.

[TBD still correct? Need to expand?]

C.5.2 Node Area

The Node Area of the product label is a container for all discipline-specific classes and attributes. These are the classes and attributes that are defined in various PDS node-level local data dictionaries (see section 4.9); they must be prefixed with the relevant node's namespace identifier.

```
<Node_Area>
  <img:Camera_Parameters>
  <img:exposure_duration>46.0</img:exposure_duration>
  <img:exposure_type>AUTO</img:exposure_type>
  <img:filter_name>L670_R670</img:filter_name>
  <img:filter_number>5</img:filter_number>
  </img:Camera_Parameters>
</Node_Area>
```

Note that it is possible to utilize classes and attributes pulled from multiple node dictionaries. In this case, a separate Node Area would be used for each node's elements.

[TBD – need to expand?]

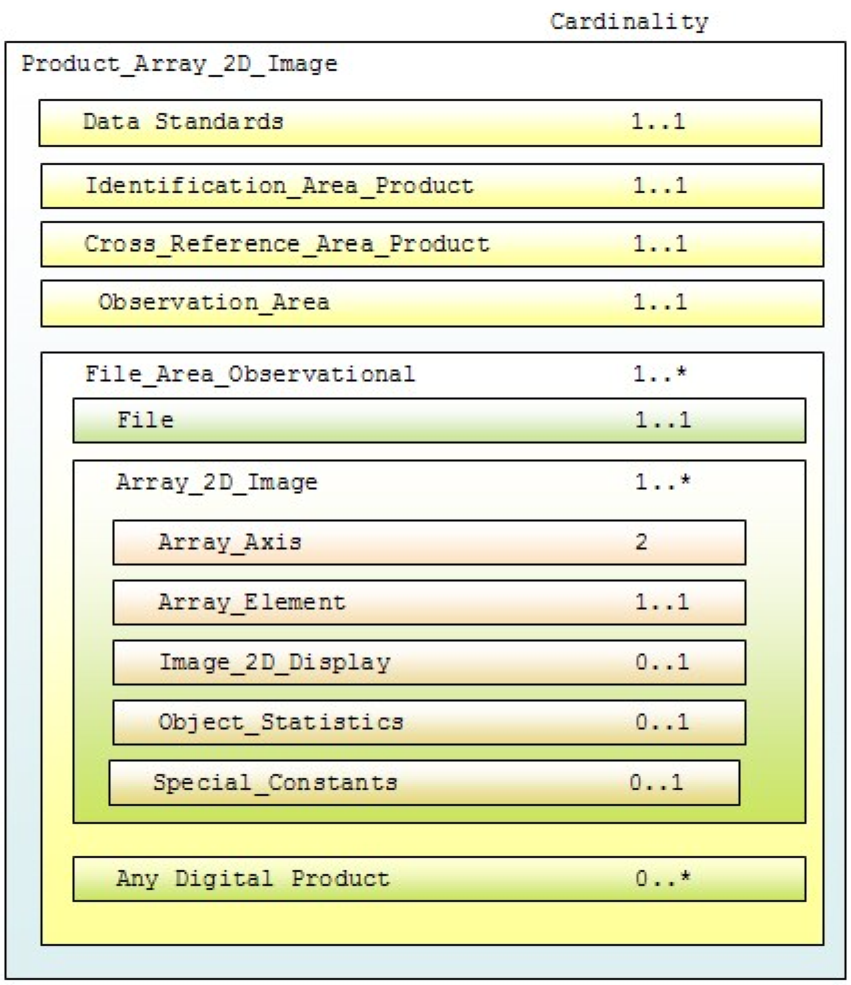
C.6 File Area

The File Area of a product XML label document is used to reference the data file(s) (i.e., the string of bits) being described by the label. The File Area is repeated for each data file.

[TBD – need all of this, including separate entries for at least array 2D image, character table, and perhaps header.]

APPENDIX D EDITING A LABEL TEMPLATE

This appendix includes a block by block description of editing each major section of an observational product and how each section of the label template could be edited to tailor the template to be more specific with respect to the nature of your particular archive.



D.1 Data Standards

The Data_Standards block in the template looks like this:

```
    <Data_Standards>
      <dd_version_id>dd_version_id0</dd_version_id>
      <std_ref_version_id>version_id0</std_ref_version_id>
    </Data_Standards>
```

The dd_version_id attribute is set to the version identifier for the data dictionary that was currently in use when the archive was produced.

The std_ref_version_id is set to the version of the Standards Reference that was currently in use when the archive was produced.

D.2 Identification Area

The Identification Area block in the template looks something like this:

```
<Identification_Area_Product>
  <logical_identifier>logical_identifier0</logical_identifier>
  <version_id>version_id0</version_id>
  <product_class>Product_Table_Character</product_class>
  <title>title0</title>
  <alternate_title>alternate_title0</alternate_title>
  <alternate_id>alternate_id0</alternate_id>
  <last_modification_date_time>lmdtime0
  </last_modification_date_time>
  <type>product_subclass0</type>
  <Subject_Area>
      ...
  </Subject_Area>
</Identification_Area_Product>
```

Most of the XML elements have placeholder values inserted (e.g., logical_identifier0 for pds:logical_identifier). The exception is product_class which has the actual value, Product_Table_Character, rather than a placeholder. This is because all of our labels will have the same value, so we set the value explicitly in the edited schema using fixed="Product_Table_Character" in the product class XML element.

D.3 Cross Reference Area

Populating entries in the cross reference area of a product XML label document is remarkably straightforward considering the apparent complexity when viewing the corresponding section of the schema. There are three types of entries that can appear in this section and all three refer to information separate from this product, but related to / referenced from this product.

```
                         | –  Bibliographic Reference
                         |
Cross Reference Area —   | –  Observing System
                         |
                         | –  Reference Entry
```

- Bibliographic References provide references to documents, relevant to the product. Examples include references to a journal article describing the detailed processing used to produce the product.
- Observing System references one or more subsystems (such as spacecraft – instrument or observatory – telescope - detector combinations.
- Reference Entries provide references to products, relevant to the products, which are archived in the PDS. Examples include references to browse products, calibration products, document products, collection products, etc.

D.4 Observation Area

[TBD]

- Mission Area
- Node Area

D.5 File Area

The File Area of a product XML label document is used to reference the data file(s) (i.e., the string of bits) being described by the label. The File Area is repeated for each data file. Most XML labels will reference a single data file and therefore there would be a single File Area class.

There are multiple instances of the File Area class (e.g., File_Area_Observational, File_Area_Text, File_Area_XML_Schema). Each instance has very little variation in terms of the attributes used in the class. The principle variations are in whether (or not) the offset attribute is optional or required and whether (or not) the file_specification_name attribute is present or absent.

For the purposes of our discussion, we will use the File_Area_Observational – since this is the class that you will use in your observational science products.

The File Area Observational block in the template looks something like this:

```
<File_Area_Observational>
    <File>
        <local_identifier>local_identifier3</local_identifier>
        <comment>comment2</comment>
        <creation_date_time>creation_date_time0</creation_date_time>
        <file_name>file_name0</file_name>
        <file_size>0</file_size>
        <maximum_record_bytes>50</maximum_record_bytes>
        <md5_checksum>00000000000000000000000000000000</md5_checksum>
        <records>50</records>
    </File>
        .
        .
        .
</File_Area_Observational>
```

All of the XML elements have placeholder values inserted (e.g., logical_identifier2 for pds:logical_identifier). Populating entries in the file area observational of a product XML label document is remarkably straightforward. The file block consists of a number of optional and required attributes that describe the data file.

- The local identifier attribute, an optional attribute, provides the name of the local object being described; the value must be unique within the XML label.
- The comment attribute, an optional attribute, provides a brief description of the comment.
- The creation date time attribute, an optional attribute, provides the date/time when the file was created, either in YMD or DOY format.
- The file name attribute, a required attribute, provides the name of the file being described.
- The file size attribute, an optional attribute, provides the size (in bytes) of the file.
- The max record bytes attributes, an optional attribute, provides the size (in bytes) of the maximum record within the file. For example, in a character table having a fixed-length 78 byte record structure, the max_record_bytes is set to 78 (which is inclusive of the record delimiters).
- The md5 checksum attribute, an optional attribute, provides the md5 checksum of the file.
- The records attribute, am optional attribute, provides the number of records in the file.

The next block contains the reference to the product components (objects) (e.g., table character, array 2d image, etc) being described by the XML label.

For example, if the data file contains a header, and two character tables, then the File_Area_Observational block would look like:

```
<File_Area_Observational>
    <File>
        <local_identifier>local_identifier2</local_identifier>
        <comment>comment1</comment>
        <creation_date_time>creation_date_time0</creation_date_time>
        <file_name>file_name0</file_name>
        <file_size>0</file_size>
        <max_record_bytes>50</max_record_bytes>
        <md5_checksum>00000000000000000000000000000000</md5_checksum>
        <records>50</records>
    </File>
    <Header>
        <local_identifier>local_identifier10</local_identifier>
        .
        .
        .
    </Header>
    <Table_Character base_class="Table_Base">
        <local_identifier>local_identifier16</local_identifier>
        .
        .
        .
    </Table_Character>
   <Table_Character base_class="Table_Base">
        <local_identifier>local_identifier16</local_identifier>
        .
        .
        .
    </Table_Character>
</File_Area_Observational>
```

D.5.1 File Area - offset

Note that additional metadata is required for each of the above product components to ensure the structure of each is well understood (e.g., offset). Using the header object as an example.

```
<Header>
    <local_identifier>local_identifier11</local_identifier>
    <comment>comment9</comment>
    <name>name21</name>
    <description>description5</description>
    <bytes>0</bytes>
    <encoding_type>CHARACTER</encoding_type>
    <external_standard_id>FITS</external_standard_id>
    <offset unit_of_measure_type="byte">50</offset>
</Header>
```

1. The number of bytes has to be set to indicate the total number of bytes used by the header object.
2. The encoding type has to set to indicate the header is using either character or binary encoding.
3. The offset has to be set to indicate the starting location (in bytes) of the header object, which in this case the header has a 0 byte offset. Note that the offset for the 1st

character table would be set to the bytes used by the header object + 1. The offset for the 2nd character table would be set to the total bytes used by the header and 1st character table + 1.

D.5.2 File Area – data file location

The data files referenced by a product label must either be co-located with the label, or in a subdirectory of the directory containing the product label. Note that some product labels require the data file(s) to be co-located with the product label.

If the product label includes the file_name attribute, then the data file(s) must be co-located with the product label – as mechanically there isn't a way to specify the location of the files relative to the product label.

When the product label includes the use of the directory_path_name attribute, the product label may either be co-located with the data files, or in a subdirectory of the directory containing the product label. The value specified in the directory_path_name attribute is always relative to the location of the product label. If the product label and the data file are co-located, then the value is simply the name of the file:

```
<directory_path_name>filename.tab</directory_path_name>
```

If the data file(s) were located in the "test" subdirectory, then the value would be:

```
<directory_path_name>test/filename.tab</directory_path_name>
```

Note that directory paths are always specified in UNIX system format (i.e., the forward slash ("/") is used to denote directories and to delineate subdirectories).

Additional information on "The File Area" and how it is used within the archive can be found in the Standards Reference[2]. For specific uses of the File Area instances, you should consult the Data Dictionary[3a,b].