Data Providers' Handbook

Archiving Guide to the PDS4 Data Standards

DRAFT

Data Design Working Group

March 2011

Version 0.3

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 Chapter 3, except Chapter 2	Simpson
0.22.2	2010-09-24	Added comments from Mitch Gordon	Simpson
0.22.2	2010-10-01	Significant edits to Chapter 4	Simpson
0.22.3	2010-10-25	Significant edits to Chapters 1,3-6	Simpson
0.3	2011-04-15	Significant edits addressing Build 1b comments	M.Gordon etal

TABLE OF CONTENTS

1.0 INTRODUCTION

First time readers should consult the "Introduction to the PDS4 Document Set" (IPDS) for an overview of the documents that collectively describe PDS4. The IPDS lists the documents roughly from most general to most specific and potentially in the order that you should read them, and provides a brief description of each.

The Introduction to the PDS4 Document Set can be found at:

http://pds.jpl.nasa.gov/build1creview/DocSetIntro-20110309.pdf

1.1 Purpose

While, the PDS4 Standards Reference remains the definitive source for ensuring data meet the PDS4 archive criteria, the DPH functions more in the capacity of a tutor/coach to provide information and examples to guide data providers in the design and preparation of data to be archived with the PDS.

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 Document Notation

[TBD2]

1.4 Applicable Documents

1.4.1 Controlling Document

[1] Planetary Data System (PDS) PDS4 Information Model Specification, Version 0.3.0.0.e

1.4.2 Reference Documents

[2] Planetary Data System Standards Reference, April 2011, Version 4.0.3 (beta), JPL D-7669, Part 2

[3] Planetary Science Data Dictionary Document, March 2011, JPL D-7116 Rev. X.

[4] PDS4 Data Standards Concepts, March 22, 2011.

[5] Glossary of PDS4 Terms, March, 2011, Version 2011-03-24 (v110324).

[6] PDS4 Data Dictionary Tutorial, April 2011.

2.0 A FEW KEY TERMS

This section provides a review of a few key terms you will encounter early in this document. This is not intended to be either comprehensive or exhaustive. We assume you have read the Concepts Document and at least previewed the Glossary of PDS4 Terms.

2.1 Product

A product is one or more tagged objects (digital, non-digital, or both) grouped together and having a single, PDS-unique identifier. In the PDS4 implementation, the descriptions are combined into a single XML label.

2.2 Basic Products

The simplest product in PDS4; one or more data objects (and their description objects), which constitute (typically) a single observation, document, etc. The only PDS4 products that are *not* basic products are Product_Collection and Product_Bundle. Every basic product must be a primary member of one (and only one) collection. Basic products may be secondary members of any number of collections

2.2.1 Primary/Secondary Membership

A *basic product* is a primary member of the collection within which it first entered into PDS4. Every basic product must be a primary member of one (and only one) collection. Basic products may be secondary members of any number of collections.

The type of membership is given in the collection inventory.

2.3 Collections

A list of basic products, all of which are closely related in some way. The list is referred as the collection inventory. A collection is itself a product (because it is simply a list, with its label); but it is not a *basic* product.

2.4 Bundles

A list of collections. The list is referred as the bundle inventory. For example, a bundle could list a collection of raw data obtained by one instrument during a mission lifetime, a collection of the calibration products associated with the instrument, and a collection of all documentation relevant to those collections.

2.5 Labels

The aggregation of one or more description objects (composed of PDS classes and attributes) such that the aggregation describes a single PDS product. In the PDS4 implementation, labels are constructed using XML and are always derived from a PDS schema.

A 'generic' schema has been established for each type of anticipated PDS4 product; these can be tailored by data providers to become 'specific' schemas for their archives. See

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

for the current set of generic schemata available for general use.

2.5.1 XML Terms

A few words have different meanings depending on the communities in which they are used. We have adopted modifiers to help distinguish between multiple uses. For example, 'attribute' is widely used in PDS and in XML where it has a different meaning. In this document we use 'PDS attribute' and 'XML attribute' to establish the context.

XML element: An XML structure that begins with <tag>, contains 'content', and ends with </tag>. For example, "<date>2009</date>" is an XML element that establishes the date as 2009.

XML schema: The definition of an XML document, specifying required and optional XML elements, their order, and parent-child relationships.

2.5.2 XML Editors

Use of an XML editor simplifies construction and validation of schemata and labels. Two which have been popular during development of PDS4 are oXygen (http://www.oxygenxml.com) and Eclipse (http://www.eclipse.org/downloads/download.php?file=/eclipse/downloads/drops/R-3.5.2-201002111343/eclipse-SDK-3.5.2-win32-x86_64.zip).

2.6 Identifiers

A unique character string by which a product, object, or other entity may be identified and located. Identifiers can be global, in which case they are unique across all of PDS (and its federation partners). A local identifier must be unique within a label.

2.6.1 Logical Identifier (LID)

An identifier which is unique across PDS. Every PDS4 product must have a LID which is provided in the product label.

2.6.2 Version Identifier (VID)

A version identifier with specific formation and incrementation rules.

2.6.3 LIDVID

The concatenation of a LID and VID. Provides a globally unique identifier for every version of every product.

Among other things, basic product LIds or LIDVIDs are used in collection inventories to identify the collection members, and collection LIDs or LIDVIDs are used in bundle inventories to identify the bundle members

2.6.4 Local Identifier

Local identifiers may be used within a label so that product components (objects) can be easily located. Local identifiers are not valid outside the label for which they are defined.

3.0 WHAT TO DO

Here is a fairly high level outline of the steps that need to be taken in order to develop an archive for PDS ingestion. In the next chapter we begin the discussion of how to do these things. While the implied order is intentional, many of the steps overlap and will be pursued in parallel, and many of these tasks are iterative. Consequently, the process is not nearly as linear as the simple outline implies.

So where to begin? Start by thinking about bundles. A bundle is an aggregation of collections which have been organized together because of a common ‘theme’.

Determine what that the theme is for your data. One typical theme is to aggregate all of the observational data obtained by a single instrument on a large mission along with its supporting documentation and ancillary files. Another theme might be to aggregate all of

the data obtained by a team on a multi-year observing campaign along with its supporting documentation and ancillary files. Obviously there are lots of potential themes.

PDS requires comprehensive documentation and support materials (e.g., calibration documents and products) for its archives. Consider how you will handle that material. Much of it will be included in the same bundle as the data; however, some of that material may be submitted in separate bundles and referred to from products in your bundle.

Now we can look at the steps ahead.

- Outline the Bundle
    - Determine which collections will be in your bundle.
    - Determine the documentation you will need to produce.

- Design LIDs and VIDs

- Design Basic Products – We will need to repeat this for the basic products in each collection. Pick one collection to begin with.
    - Remember, a basic product is one or more data objects and a label file containing the description(s) of the data object(s).

    - Design the data object(s)
        - What type of product? (character table, 3D image, etc.)
        - Determine the organization of the data (number of table columns, image characteristics, etc.).
        - Determine the appropriate storage structure – this is highly constrained in PDS4
        - Really want to be in close coordination with the discipline node by now. Really.

    - Design the label
        - Obtain an appropriate starting schema from your discipline node
        - Determine if you need a Mission Dictionary; if so start it.
        - Edit and validate the schema
        - Generate and validate an XML template of the label.

    - Determine what additional documentation you need to provide.
        - Begin preparing the additional documentation.

- Design the Collection – We need to repeat this for each collection
    - Obtain an appropriate schema from your discipline node.

    - Edit the schema.

    - Generate an XML Template.

    - Determine what additional documentation you need to provide.
        - Begin preparing the additional documentation.

- Design the Bundle
    - Obtain an appropriate schema from your discipline node.
    - Edit the schema
    - Produce the XML file – this contains both the label contents and the inventory of the bundle.
    - Prepare the additional documentation (e.g., readme, errata)

- Design the Delivery Package

- Generate the Basic Products – Assumes you have the data.
    - Produce the basic product data files.
    - Generate the basic product labels and validate.

- Populate the collection inventories.

- Prepare the delivery package and submit to PDS.

4.0 HOW TO DO IT

4.1 Using the Data Dictionary.

Throughout this chapter we will refer to the data dictionary and in many cases look in detail at dictionary entries.

Some notes about searching the dictionary. – There are issues which we are still trying to resolve.

1. We are still working on the contents and structure of the abridged data dictionary in order to improve its ease of use.

2. Strategies for finding an entry for a specific class or attribute:

    a. Start at the hyperlinked indices – at the end of the document are separate indices for attributes and classes. However at the time of this writing the class index is no comprehensive.

    b. If the indices don't resolve your search, enter the specific product (e.g., Product_Table_Character) you are working on in the Acrobat search box. This will give a small number of hits – typically less than 10. Find the primary entry for the product and scroll to the class or attribute you need.

    c. Enter the class or attribute you are working on in the Acrobat search box. This will produce a lot of hits. Many of these will give the definition. However, classes which do not appear in the end of dictionary index generally have context dependant characteristics typically for the number of occurrences, whether or not the attribute is optional, and the enumerated list of values if it has one. You must check carefully the context of the entry you find (e.g., if it is listed within a Product, is it the correct product?).

    d. Contact your PDS discipline node. Asking for help may save you time and will let us know what still needs to be improved.

4.2 Construction of Product Identifiers

Every product label contains an identifier which is unique. 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.

4.2.1 LID Construction – General Concepts

Here are some general rules:

- **LIDs must be unique across PDS**
- **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 are in Chapter 7 of the PDS4 Standards Reference.

4.2.2 Constructing LIDs

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

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

- LIDs for bundles are constructed by appending a bundle specific segment to 'urn:nasa:pds'.

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

  Since all PDS bundle LIDs are constructed this way, the bundle segment must be unique across PDS.

- LIDs for collections are constructed by appending a collection segment to the parent bundle's LID.

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

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

- LIDs for basic products are constructed by appending a product segment to the parent bundle's LID.

  Collection LID =
  urn:nasa:pds:<bundle segment><collection segment><product segment>

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

- Some examples are in order. Suppose we have a hypothetical, new archive being prepared by an instrument team on a planetary mission.

The spacecraft is the first in a new generation of NASA exploratory vehicles and the instrument is a high precision photon counter. The mission is to orbit Venus, following a modest duration cruise phase which includes Mercury and Earth flybys. Here are the names and abbreviations.

spacecraft	Super SpaceCraft 01	SSC01
instrument	High Resolution Photon Counter	HiResPC
cruise phase	Cruise, Mercury, Earth phase	Cr-Me-Ea
Venus orbit phase	Venus Encounter	Ven

The instrument team plans to submit two archive bundles to PDS, one for the cruise phase data, and one for the Venus encounter data.

Each bundle will have several collections. We'll consider four of them for the purposes of our discussion. Those are Browse, Data_Raw, Data_Derived, and Document. In this case, the browse collection contains footprint diagrams of the field of view of the instrument for each observation.

The team decides to use the spacecraft clock count at the start of each observation as the primary reference for the observation. This is also used as the filename root and the data product segment of the LID.

Each bundle's document collection will have several documents unique to that bundle. Each will have its own errata.txt document which we'll use in the example.

This is all we need to start designing LIDs.

Cruise Phase
Bundle

urn:nasa:pds:ssc01.HiResPC.C-E-M [bundle segment = spacecraft.inst.phase]

Collections
urn:nasa:pds:ssc01.HiResPC.C-E-M:Browse
urn:nasa:pds:ssc01.HiResPC.C-E-M:Data_Raw
urn:nasa:pds:ssc01.HiResPC.C-E-M:Data_Derived
urn:nasa:pds:ssc01.HiResPC.C-E-M:Document

Products data products for sclock = 31234567
urn:nasa:pds:ssc01.HiResPC.C-E-M:Browse
urn:nasa:pds:ssc01.HiResPC.C-E-M:Data_Raw
urn:nasa:pds:ssc01.HiResPC.C-E-M:Data_Derived
urn:nasa:pds:ssc01.HiResPC.C-E-M:Document:errata

Venus Encounter Phase
Bundle [bundle segment = spacecraft.inst.phase]
urn:nasa:pds:ssc01.HiResPC.Ven

Collections
urn:nasa:pds:ssc01.HiResPC.Ven:Browse
urn:nasa:pds:ssc01.HiResPC.Ven:Data_Raw
urn:nasa:pds:ssc01.HiResPC.Ven:Data_Derived
urn:nasa:pds:ssc01.HiResPC.Ven:Document

Products data products for sclock = 76543213
urn:nasa:pds:ssc01.HiResPC.Ven:Browse:76543213
urn:nasa:pds:ssc01.HiResPC.Ven:Data_Raw:76543213
urn:nasa:pds:ssc01.HiResPC.Ven:Data_Derived:76543213
urn:nasa:pds:ssc01.HiResPC.Ven:Document:errata

4.2.3 VID Construction

Version IDs are used for all types of products, including standard 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.

- Whenever the major number is incremented, the minor number is re-set to zero.
- The minor portion of the version is not pre-padded with zeros; it is simply incremented as an integer. Thus, “1.1” and “1.10” are different versions; “1.01” is invalid.

Our sample VIDs (bundle, collection, and product) are 1.0 for the initial submission to PDS.

4.2.4 LIDVID Construction

Concatenate the LID and VID using a double colon as the connector. Here are sample LIDVIDs based the LIDs in Section 4.1.2:

Cruise Phase

Bundle

urn:nasa:pds:ssc01.HiResPC.C-E-M::1.0

Collections

urn:nasa:pds:ssc01.HiResPC.C-E-M:Browse::1.0
urn:nasa:pds:ssc01.HiResPC.C-E-M:Data_Raw::1.0
urn:nasa:pds:ssc01.HiResPC.C-E-M:Data_Derived::1.0
urn:nasa:pds:ssc01.HiResPC.C-E-M:Document::1.0

Products data products for sclock = 31234567

urn:nasa:pds:ssc01.HiResPC.C-E-M:Browse::1.0
urn:nasa:pds:ssc01.HiResPC.C-E-M:Data_Raw::1.0
urn:nasa:pds:ssc01.HiResPC.C-E-M:Data_Derived::1.0
urn:nasa:pds:ssc01.HiResPC.C-E-M:Document:errata::1.0

Venus Encounter Phase

Bundle

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

Collections

urn:nasa:pds:ssc01.HiResPC.Ven:Browse::1.0
urn:nasa:pds:ssc01.HiResPC.Ven:Data_Raw::1.0
urn:nasa:pds:ssc01.HiResPC.Ven:Data_Derived::1.0
urn:nasa:pds:ssc01.HiResPC.Ven:Document::1.0

Products data products for sclock = 76543213

urn:nasa:pds:ssc01.HiResPC.Ven:Browse:76543213::1.0

urn:nasa:pds:ssc01.HiResPC.Ven:Data_Raw:76543213::1.0
urn:nasa:pds:ssc01.HiResPC.Ven:Data_Derived:76543213::1.0
urn:nasa:pds:ssc01.HiResPC.Ven:Document:errata::1.0

4.2.5 LIDVIDs – The Next Step

When you have constructed draft Bundle LIDs and LIDVIDs contact your PDS lead node to verify they are unique and conform to the requirements.

4.3 Developing Basic Products

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.

4.3.1 Basic Product Label Organization (other than Document Products)

There are four major blocks of information (areas) in a typical basic product label. Each area contains one or more classes each of which may have several attributes. The four areas are Identification Area, Cross Reference Area, Observation Area, and File Area.

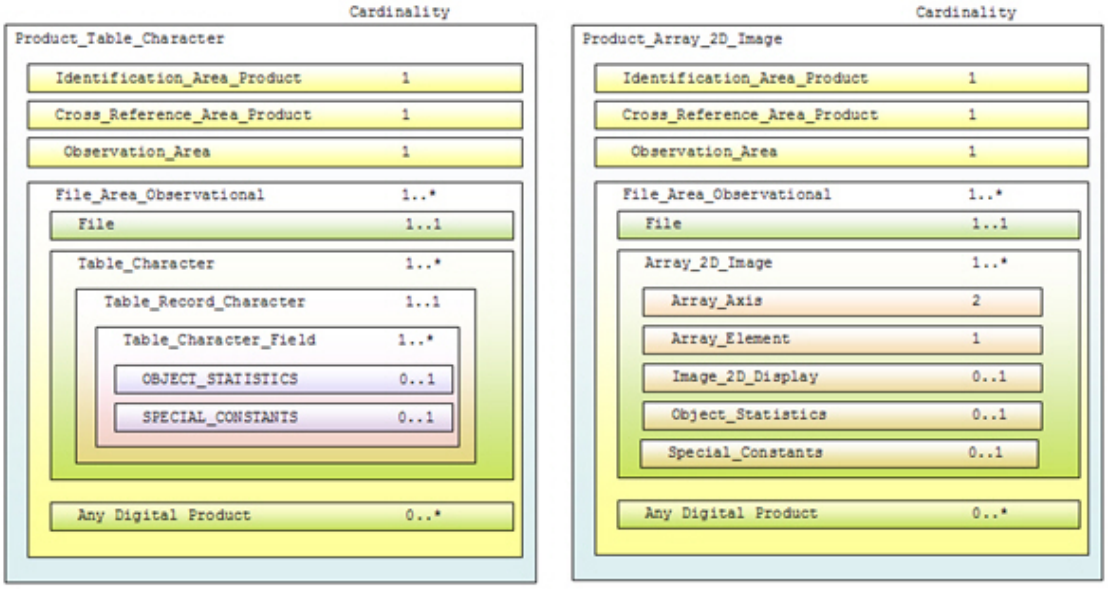


Figure 4-1. Basic Product (other than Document) Label Organization

Figure 4-1 shows 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 4-1 only reflects the PDS specific content of the labels. There will be several lines of text preceding the information depicted here which are overhead associated with the use of XML.

4.3.2 Schema

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

In consultation with your PDS discipline node determine which schema(s) you will need. The discipline node will take the appropriate generic schema(s) 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 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.

If you are developing a collection with more than a few products, you will want to automate label generation as much as possible. PDS is working on tools to aid in this.

There are at least two approaches:

a) use the specific schema as input to your pipeline software, or

b) use an XML label template as input to the pipeline software. One feature of an XML editor is the ability to generate such a template by pushing a button (after first setting some options).

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

The template looks like the final label except that values which vary from one label to the next are represented by placeholders which the pipeline software will replace.

In a schema you may indicate the number of times an entry will appear (e.g., you can specify the number of fields in one line of a table). However, you may not actually replicate the entry in the schema. This restriction does not apply to XML files.

In this chapter, 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 this chapter including the section on template generation.

In this guide, 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.

We begin with the generic version of the product table character schema, Product_Table_Character_*.xsd, where * gives a version number.

A small set of examples has been provided [TBD1] with this document. The set are based on the hypothetical mission described in the section on Identifiers and includes:

- the generic schema Product_Table_Character
- three schema files identified as ‘include files’ in the schema
- a specific version with the edits described in the this document
- an XML template generated from the specific schema
- a modified version of the template with the edits described in the this document
- one label file produced from the template

At this point you should open Product_Table_Character_*.xsd in your XML editor in text view, or in your favorite XML aware editor.

4.3.2.1 Reading a PDS Schema

Place the schema you will be editing into a working directory. You will also need any included schema in the same directory. You can identify these by finding the line or lines beginning “<xsd:include schemaLocation=...” in the top portion of the schema. The file name of the included file follows the equal sign. These typically are schema files for Attribute_Types, Base_Types, and/or Class_Types.

Make a copy of the schema you intend to modify and give it an appropriate name (.e.g., insert the abbreviation for your spacecraft and instrument into the original name). This copy is the version you will be editing. Open it in an editor.

Using an XML specific editor such as Oxygen or Eclipse adds substantial capability and will make the task less time consuming. When you first open the schema, 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 an XML aware editor such as BBEdit or UltraEdit, but such editors do not provide validation checking. For more information on XML editing and editors, see [TBD2].

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

[TBD2] – provide specifics – when, why, how

The actual 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 more than once). 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).

4.3.2.2 Permitted Modifications

When you modify a 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.
b) You may delete an optional XML element.
c) 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.
d) 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.
e) During the initial tailoring, the PDS node staff may insert additional classes in the Node Area (a subsection within the Observation_Area).
f) As the data provider, you may insert additional classes in the Mission Area (a subsection within the Observation_Area).

The schema has a lot of XML elements. We already know what some of them represent. For the others we can search the abridged data dictionary to determine what they are and what restrictions, if any, are placed on their values. We can use this information coupled with the constraints above to determine what modifications to make to the schema.

There are several approaches to handling optional XML elements. Which you choose to use will be based at least in part on your pipeline design.

In most cases when we make a specific schema (our current task) it will support one set of products. All of the labels will be structurally the same, and the values of some of the entries will differ from one to the next (e.g., all refer to the same instrument, but each has a different start time). In this situation, as we go through the schema we can determine whether or not a particular optional XML element actually will be used, and we only need to distinguish between the two options – will or won't be used. There are three approaches that may be used.

a) set minOccurs to 1 for the optional XML elements you intend to use, thus making them required as far as the XML editor is concerned. Then you set the editor preferences to ignore optional elements when generating the XML template. See the users manual for your editor for the specifics on doing this.
b) Alternatively you can leave minOccurs equal to 0 for the optional XML elements you intend to use; delete the other optional XML elements, and set the editor preferences to include optional elements when generating the XML template.
c) Finally you may set minOccurs to 1 for the optional XML elements you intend to use, and delete the other optional XML elements.

In more complicated instances, one specific schema will support related products and some optional XML elements may be included in the labels of only a portion of the products. In this situation, the pipeline will need to know the conditions for including or removing these particular elements. Perhaps the easiest approach is a variation on c) above.

- Set minOccurs to 1 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.
- Leave "minOccurs =0" for the optional XML elements you intend to use only for some subset of the products.

4.3.2.2.1 Permitted Modifications – The Next Step

Regardless of how you choose to proceed, once you have made a few edits to a schema, we strongly encourage you to show them to your PDS node to ensure you are on the right track before you invest a lot of time and effort.

4.3.2.3 Specific Entries and Sample Edits

Let's start at the top of the actual label contents of the schema and go section by section. In the following discussion with regard to optional XML elements, we will assume the

simple case in which all product labels will use the same XML elements and we will employ option c) above.

The gross outline of the schema is simple, and is given in the first section, Product_Table_Character_Type. As we saw earlier, this section indicates that the schema has four major Areas:

Product_Table_Character_Type

- Identification_Area_Product
- Cross_Reference_Area_Product
- Observation_Area
- File_Area_Observational

We begin by briefly revisiting the **Product_Table_Character_Type** section in the schema which we previously examined in the general discussion about reading a schema (Section 4.2.2.1) .

Earlier we determined that we cannot modify the entries for the first three XML elements (Identification_Area_Product, Cross_Reference_Area_Product, Observation_Area) in this section. However we can consider modifying the XML element, File_Area_Observational which is the area which describes the file or files containing the observational data. We (as the ones developing the archive) know that each of our character table will consist of one data file (and its associated label file of course). Consequently in the XML element, File_Area_Observational, we can change the value for maxOccurs from “unbounded” to 1.

Change from

```
 <xsd:element name="File_Area_Observational"
  type="pds:File_Area_Observational_Type" minOccurs="1" maxOccurs="unbounded">
```

to

```
 <xsd:element name="File_Area_Observational"
  type="pds:File_Area_Observational_Type" minOccurs="1" maxOccurs="1">
```

4.3.2.3.1 Specific Entries and Sample Edits – An Outline

It will be useful to provide an outline of the remainder of the schema to guide the remaining discussion. We need to discuss the organization and establish a convention before we can actually write the outline.

Remember if the value of name is given in camel case, the item is a PDS class. Each of the four areas listed in the XML elements in this section represents a PDS class, so it will contain multiple PDS attributes, and/or subclasses. Some of the embedded subclasses will contain additional subclasses. This embedding complicates writing the outline.

Anytime we have an XML element which is a PDS class, we will also have a corresponding "_Type" block that expands the class to show its contents. For example, the first XML element in Product_Table_Character_Type section is "Identification_Area_Product". The expansion of this area is in the block titled "Identification_Area_Product_Type". Some of these "*_Type" blocks are in the schema we are working on, but in order to reduce the clutter in the schema, many of these "*_Type" blocks are in one of the included schema, specifically Class_Types.xsd.

The first occurrence of an embedded subclass is Subject_Area which is in the Identification_Area. In the "Identification_Area_Product_Type" block, the entry is "Subject_Area". In the schema, immediately following the "Identification_Area_Product_Type" is the "Subject_Area_Type" block.

In our outline we only show one entry for "Subject_Area" and expand it in place (implicitly in the schema or in the included schema, Class_Types.xsd, there for each class there is a corresponding "*_Types" block). In addition to merging the "*_Type" blocks in our outline, for the sake of brevity we indicate the existence of PDS attributes without listing them explicitly. We retain the case convention for names to distinguish between classes and attributes.

With these conventions in mind, the outline of our schema, beginning below the top "Product_Table_Character_Type" section looks like:

- Identification_Area_Product
  - several attributes
  - Subject_Area
    - several attributes
  - Name_Resolution
    - several attributes

- Cross_Reference_Area_Product
  - Bibliographic_Reference
    - several attributes
  - Observing_System
    - several attributes
    - Observing_System_Component
      - one attribute
      - Observing_System_Reference_Entry
        - several attributes
  - Product_Reference_Entry
    - several attributes

- Observation_Area

    - several attributes
    - Mission_Area
        - xsd:any
    - Node_Area
        - xsd:any

- File_Area_Observational
    - several attributes
    - File
        - several attributes
    - Table_Character
        - several attributes
    - Any_Digital_Object*
        - several attributes

* Any_Digital_Object in this context is just a placeholder. We will resolve this when we reach this section in the discussion.

4.3.2.3.2 The Schema – Block by Block

In order to write the remaining portion of this chapter, in addition to having the schema opened in an editor, we opened the data dictionary. To begin, we searched on Product_Table_Character; selected the entry with the full description of Product_Table_Character; scrolled to the Identification_Area_Product portion of that entry. You will want to do the same.

- 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 not 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).

  In our simple example title will have the same value for all products generated with the schema we are producing, so we can insert the value, or leave it as is and insert the value in the pipeline. In our example file we use the "fixed=" approach and set the value to?

- 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 three different ways people refer to the products you are labeling with this schema, one of those is the title, and the other two are alternate titles. 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. The function of this attribute is under review. so we deleted the entry.

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

  We're going to use this as an example to demonstrate using of the data dictionary. We'll do this periodically to illustrate other aspects of using the dictionary. Since we have the dictionary open to the Identification_Area_Product section of Product_Table_Character, we just click on the entry

attribute:**last_modification_date_time**

From the abridged dictionary, here is the full entry the link takes us to:

last_modification_date_time
steward: **pds**
name space id: **pds:**
version: **0.3.0.0.e**

- 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.

Since this entry is in the Identification_Area, the attribute applies to the product. The product is the label file and the information objects which it describes – in our case the label and the character table which it describes. The value we should use is the last time either the label or the table was modified, so probably the time the label file was written. Consequently this value will be populated by the 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 ASCII encoding.**
character_constraint: **ASCII**
enumerated_flag: **F**
formation_rule: **yyyy-mm-ddThh:mm:ss.sss/yyyy-doyThh:mm:ss.sss**
maximum_characters: **30**
minimum_characters: **1**
xml_schema_base_type: **xsd:string**
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. Realistically we probably do not need to preserve the time the product was last modified to the nearest millisecond.

In our example, 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. In most cases, this can be truncated to the nearest day.

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

- Subject_Area – required. 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.

  Although subject area is required, 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. Probably will be used elsewhere in the schema. We removed it in our example.

  - instrument_name – optional.

- 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, Sometimes this field is not used. In our example?

- investigation_name – optional. if there is an appropriate investigation name, it goes here. In our example?

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

  Here are the entries in our example.

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

- Cross Reference Area

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 image 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.

- Bibliographic Reference – 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.

- Observing System – required. 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.

We'll refer directly to the Data Dictionary entry again, this time to illustrate 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.

Here is the dictionary entry for Observing_System found under Product_Table_Character:

Observing_System - Occurs 1 to * Times
description: **The Observing_System class describes the entire suite used to collect the data. The suite is described in terms of one or two observing system components. The components are characterized as either a sensor subsystem or a source subsystem.**
role: **Concrete**
attribute: value: **local_identifier *value*** Optional
attribute: value: **title *value***
attribute: value: **description *value*** Optional
    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: value: **observing_system_component_type SENSOR, SOURCE**
        Observing_System_Reference_Entry - Occurs 1 to * Times
        description: **The Observing System Reference Entry class provides a product specific reference and type information about the reference. The references are to components of the observing system.**
        role: **Concrete**
        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

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 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. Now lets look at each attribute. In case you've lost track of where we are in the outline, the last bullet was:

- 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.

- Product_Reference_Entry – 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.

- Observation_Area – 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.

  In order to write this section, we first went to the data dictionary; searched on Product_Table_Character, and selected the entry with the full description of Product_Table_Character; then scrolled to the Observation_Area portion of that entry. You will want to do the same.

- comment – optional.
- 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.
- stop_date_time – required. The date and time corresponding to the end of the observation.
- local_mean_solar_time – optional.
- local_true_solar_time – optional.
- 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.
- Mission Area – optional.

  [TBD1]
- Node Area – optional.

  [TBD1]

- File Area – required.

  [TBD1]

4.3.3 Generating and Populating a Label

We use the tools provide in an XML editor to generate a label template from our schema.

The template is a .xml file which looks much more like the final label than the schema does. However, except that values which vary from one label to the next are represented by placeholders which the pipeline software will replace. Also, if we set the options carefully in the XML editor, those XML elements which we indicated recur a specific number of times (e.g., the number of fields in a single record of our table) will have been replicated that number of times. Recall that in a schema you may indicate the number of times an entry will appear (e.g., you can specify the number of fields in one line of a table). However, you may not actually replicate the entry in the schema. This restriction does not apply to .xml files.

This means the template produced by the XML editor will have essentially identical multiple entries. In most cases we will need to edit the template to insert known values for at least some of the PDS attributes in these items. For example we will edit the entry for each field in the table record class filling in the appropriate values for that column of the table.

4.3.3.1 Basic Product Label – Block by Block

4.3.3.1.1 Identification Area

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

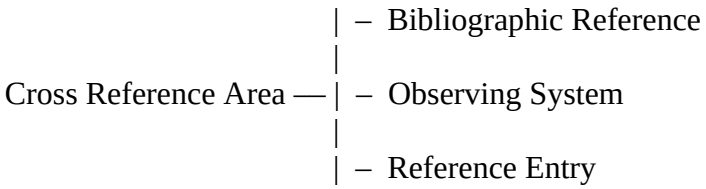
```
   <Identification_Area_Product>
     <logical_identifier>logical_identifier0</logical_identifier>
     <version_id>version_id0</version_id>
     <product_class>Product_Table_Character</product_class>
     <title>title0</title>
     <alternate_title>alternate_title0</alternate_title>
     <alternate_id>alternate_id0</alternate_id>
<last_modification_date_time>lmdtime0</last_modification_date_time>
<product_subclass>product_subclass0</product_subclass>
     <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 gave it explicitly in the edited schema using fixed="Product_Table_Character" in the product class XML element.

4.3.3.1.2 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 as plain text. There are three types of entries that can appear in this section and all three refer to information separate from this product.



- Bibliographic References provide references to documents, relevant to the product, which are not in the PDS. 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 product, which are in the PDS. Examples include references to browse products, source products, calibration documents, etc.

4.3.3.1.3 Observation Area

[TBD1]

- Mission Area
- Node Area

4.3.3.1.4 File Area Observational

The File Area Observational 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 Observational is repeated for each data file. Most XML labels will reference a single data file and therefore there would be a single File Area Observational class.

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

```
<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>
  .
  .
  .
</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, a required 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.
- 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>
```

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_identifier10</local_identifier>
    <comment>comment8</comment>
    <description>description3</description>
    <bytes>50</bytes>
    <encoding_type>BINARY</encoding_type>
    <external_standard_id>FITS</external_standard_id>
    <name>name16</name>
    <offset>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. 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.

4.3.4 Basic Product Label Organization – Document Products

[TBD1]

4.4 Developing a Collection

[TBD1]

- collection label – block by block
- member-table development
- readme (optional)
- required documents

An example can be found at the following url:

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

4.5 Developing a Bundle

[TBD1]

- bundle label – block by block
- readme (optional)
- required components
- bundle description, errata, schema, etc.
- production, versioning & validation, (pipeline if accumulating)

4.6 Developing a Delivery Package

[TBD1]

- directory structure
- manifest & checksums
- container

This section describes the process of transferring products within and external to the PDS.

4.7 Product Transfers

Transfers of products from point to point within and external to the PDS can include packaging mechanisms like ZIP, gzip, tar, physical media such as thumb drives, external hard drives, etc. The actual transfer mechanism should be coordinated with the node with which you are working.

Step #1: Select, from the set of PDS4 "Generic" Product Label Schemas, the Product Zipped schema, 'Product_Zipped.xsd'.

Step #2: Download the Product_Zipped schema, and the two global dictionaries, Extended_Types.xsd and Base_Types.xsd, from:

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

Step #3: Make a copy of the "Generic" schema and save the copy as the "Specific" Product Label Schema.

Step #4: Examine the as yet unmodified "Specific" 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.

Step #5: Use the editor to tailor this schema to be more specific to the product that you want to use. The "Specific" schema represents the overall structure and format of the product. The "Specific" schema defines, in the strictest sense, the greatest latitude permissible in the validation of the product label to ensure PDS compliance.

Examples of types of "edits/restrictions" that might be appropriate with respect to the specific schema; include:

1) Restrict the set of values in the <Subject_Area>
2) Restrict the instances in the File_Area_Type to a single reference to the type of file being described (i.e., in our example we are describing a character table having fixed length records – so we would remove all instances except the reference to File_Character_Fixed).

Expect several iterations and use the assistance of your PDS representative.

Step #6: Save the edited/tailored "Specific" Product_Zipped schema.

Step #7: 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 "Specific" schema (which is an XSD file). Save the sample label.

Step #8: Examine the sample 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 #9: Now that you have a "valid" XSD and sample label, we can proceed with populating the label with the required and optional information that will define the set of products being transferred.

Validating the data product labels is where the data product schemas become invaluable. The use of XML in data product labels and in schemas provides an expedient method by which you can ensure your product labels are PDS compliant. The PDS can offer suggestions for automating the validation process; including, the use of PDS tools.

Step #10: In the <FILE_AREA>, point to the zip file. Use the <Zipped_Member_Entry> to reference each product being zipped.

Step #11: Now that you have the zipped file and the Product_Zipped.XML label that describes the contents of the zip file, you are ready to copy these to physical media such as thumb drives, external hard drives, etc.

Step #12: Negotiate the actual transfer with the node – single transfer of the entire archive or incremental deliveries.

4.8 Delivery of Accumulating Archives

[TBD2]

4.9 Developing a 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 dicationaries 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.

4.9.1 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.

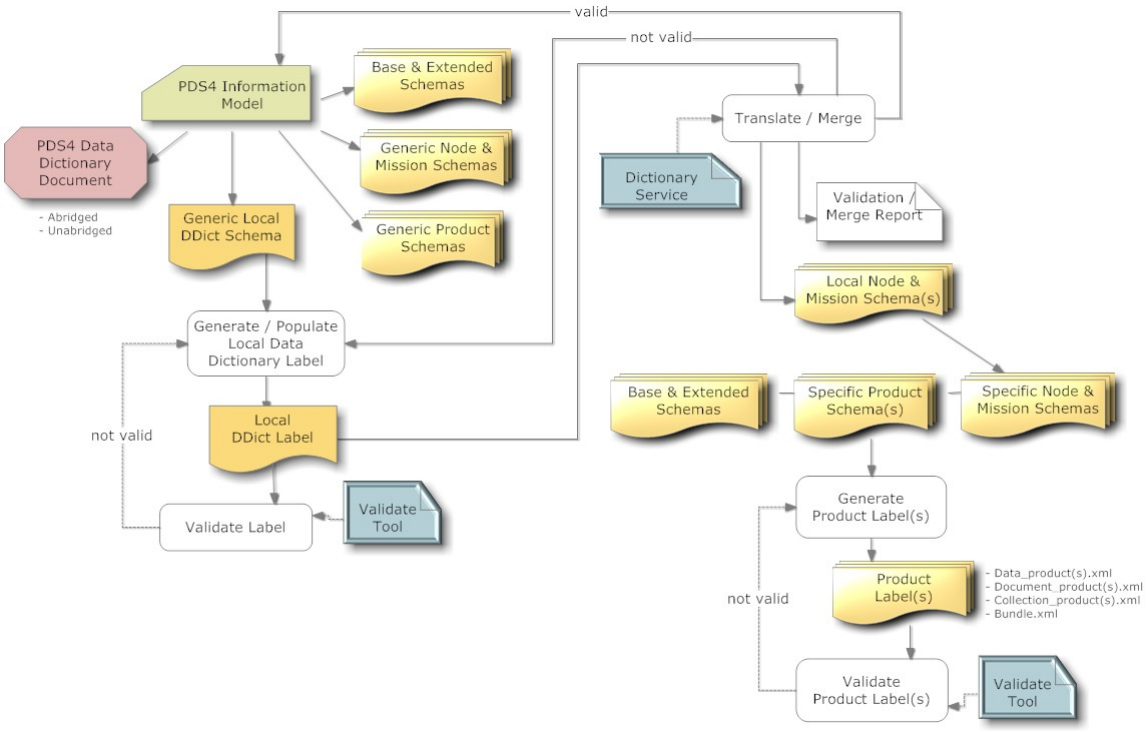


Figure 4-2 Local Data Dictionary Development Flow Diagram

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 two global dictionaries, Extended_Types.xsd and Base_Types.xsd, from:

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

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. The Validation Tool and associated documentation can be found online at:

ftp://pds.nasa.gov/pub/toplevel/2010/preparation/validate-0.1.0-bin.zip

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.

The Data Dictionary Service can be accessed online at:

http://pds.jpl.nasa.gov/tbd/...

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. See Section x.x.x for a description of the registration process.

4.10 Validating the Relationship of Schemas and Labels

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 5-3 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.

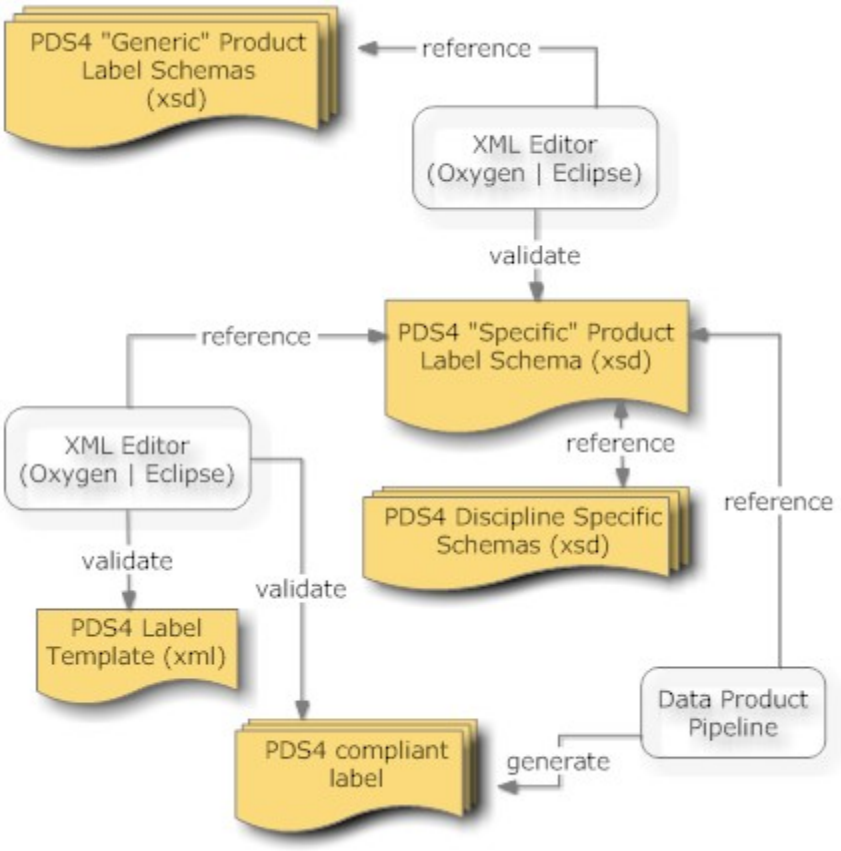


Figure 4-3. Diagram of the Validation Lifecycle of a Product Label Schema

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.

5.0 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., bundle, collections, collection inventories, aareadme, 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 (i.e., 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 was intentional so that these products could be registered with the PDS and as such searchable as "example products" and an "example archive".

5.1 PDS4 Product Examples

The set of PDS4 product examples can be found at:

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

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/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.

5.2 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.

5.2.1 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/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/dph_example_PDS3_VG2PLS/VG2PLS.zip

5.2.2 PDS4 Product Files

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

```
- Bundle_sampleArchive_20110415.xml
- Collection_sampleArchive_20110415.xml
- collection_inventory_20110415.tab
- README.TXT
- readme_20110415.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/dph_example_archive_VG2PLS/

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