Data Providers' Handbook

Archiving Guide to the PDS4 Data Standards

February 23, 2016

Version 1.4.1

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
0.3.6	2012-01-31	Updates for schema 0.7.0.0.j	M. Gordon, R. Joyner
0.3.7	2012-03-28	Updates for schema 0.7.0.0.j	M. Gordon, R. Joyner
0.3.8	2012-05-28	Updates for schema 0.8.0.0.k	M. Gordon, R. Joyner
0.3.9	2012-10-01	Updates for schema 0.3.0.0a	M. Gordon, R. Joyner
0.3.10	2013-04-01	Major changes for schema 0.3.1.0b	M. Gordon, R. Joyner, R. Simpson
1.0.0	2013-05-01	Changes for compliance with schema 1.0.0.0	M. Gordon, R. Joyner, R. Simpson
1.2.0	2014-07-01	Changes for compliance with schema 1.2.0.0	M. Gordon, R. Joyner, R. Simpson
1.3.0	2014-09-01	Changes for compliance with schema 1.3.0.0	M. Gordon, R. Joyner, R. Simpson

1.4.0	2015-04-15	Changes for compliance with schema 1.4.0.0	M. Gordon, R. Joyner, R. Simpson
1.4.0	2015-06-28	Standardized use of “schema” and “Schematron files” in various places (such as footnote 2 in Section 1.3)	Simpson
1.4.0	2015-06-28	Moved [2] and [4] to Section 1.5.2; per the CCB charter and other documents, the SR and CD are as much controlling as the IM spec.	Simpson
1.4.0	2015-06-28	Changed “supporting node” to “consulting node” per terminology in *PDS4 Concepts.*	Simpson
1.4.0	2015-06-30	Simplified Section 4 by removing references to VG2 PLS example data.	Simpson
1.4.0	2015-07-02	Rewrote Appendix D, because it didn’t work	Simpson
1.4.0	2015-07-25	Rewrote 6.1 to include product selection guidelines driven by CCB-111	Simpson
1.4.0	2015-08-10	Revised and simplified Section 7 for consistency across oXygen and Eclipse XAEs and PDS validation software.	Simpson
1.4.0	2015-08-20	Reorganized Sections 10-11 to consolidate topics and reduce repetition.	Simpson
1.4.0	2015-08-20	Streamined and reorganized Section 12. Expanded and corrected examples in 12.1. Removed most references to restricted documents from the original Section 12; then added 12.2 specifically about restricted documents.	Simpson
1.4.0	2015-08-27	Change “section” to upper case when referring to a numbered section.	Simpson
1.4.0	2015-08-28	Rewrote Section 3.0 to make it an introduction to labels and XML rather than a cookbook of how to manipulate templates (already in Section 7)	Simpson
1.4.0	2015-09-24	Added Appendices F, G, and H	Simpson
1.4.0	2015-10-06	Removed Section 14 and references thereto (the examples are outdated and have many errors). The section may be revised and returned in a later edition.	Simpson, R. Joyner
1.4.0	2015-10-06	There have been many other changes since DPH v1.3.0. The Change Log entries above should be considered representative. In many ways v1.4.0 is a new document.	Simpson. R.Joyner
1.4.1	2016-02-23	Changed date and version number on cover; updated Table of Contents.	Simpson
1.4.1	2016-02-23	Substituted period for comma at the bottom of page 13; corrected misspelling of “array” on page 18, line -6. Added missing close parenthesis in ‘blue’ text in the middle of page 23.	Simpson
1.4.1	2016-02-23	Replaced smart quotes by regular quotes in example labels.	Simpson

1.4.1	2016-02-23	Removed hyperlink notation from example labels	Simpson

TABLE OF CONTENTS

1.0 Introduction 7
1.1 Purpose 7
1.2 Audience 7
1.3 Reader Preparation 7
1.4 XML Editors 8
1.5 Applicable Documents 8
1.5.1 Controlling Documents 8
1.5.2 Document Availability 8
1.6 Other Resources 9
1.6.1 XML Schema location 9
1.6.2 PDS4 Software 9
1.6.3 PDS4 Wiki 9
1.6.4 SBN PDS4 Wiki 9
PART I. PRELIMINARIES 10
2.0 Building Blocks 10
2.1 Comments from past missions regarding archive development 10
2.2 Terminology 10
2.3 Data 11
3.0 Schema and Labels – an Overview 14
3.1 Archive Design 14
3.2 Pipeline Considerations 14
3.3 Label Design 15
3.4 Label Validation 16
3.5 Other Resources 16
PART II. THE FIRST STEPS 17
4.0 Outline the Bundle 17
4.1 Directory Organization 17
4.2 Directory and File Naming 18
4.3 Determine the Documentation Needed 19
5.0 Design Logical and Version Identifiers 20
5.1 General Concepts 20
5.2 Constructing LIDs 20
5.2.1 Examples 21
5.3 VID Construction 22
5.4 LIDVID Construction 23
5.4.1 Examples 23
5.5 LIDVIDs – The Next Step 23
PART III. BASIC PRODUCT LABELS 24
6.0 Basic Product Labels - an Overview 24
6.1 Product Selection 24
6.2 Basic Product Labels – Observational Products 25
6.2.1 Selecting the Structural Description for Observational Products 25
6.2.2 Basic Product Label Organization – Observational Products 26

6.3 Basic Product Labels – Non-Observational Products 28
6.3.1 Selecting the Appropriate Types of Non-Observational Products 29
6.3.2 Basic Product Label Organization – Non-Observational Products 29
6.4 Aggregate Product Labels 29
7.0 Label Template Creation AND Editing 30
7.1 Label Template Editing — Body 30
7.1.1 Presence or Absence of a Class or Attribute 31
7.1.2 Ensuring the Presence of a Class or Attribute 31
7.1.3 Repeating a Class or Attribute 32
7.1.4 Setting the Value of an Attribute (Fixed and Variable) 33
7.2 Label Template Editing — XML Prolog and Root Tag 34
7.2.1 XML Declaration Statement 35
7.2.2 Schematron References 35
7.2.3 Catalog Files 36
7.2.4 Root Tag – Product Type and Namespace Specifications 36
7.2.5 Root Tag – Schema Location 37
7.3 XML Label – The End Tag 38
7.4 Next Steps 38
PART IV. COLLECTIONS, BUNDLES, DELIVERY PACKAGES 39
8.0 Collections 39
8.1 Members of a Collection 39
8.2 Collection Inventory 39
8.3 Generating and Populating an Inventory Label 40
8.3.1 Collection Area 40
8.3.2 Citation Information 40
8.3.3 Context Area 41
8.3.4 File_Area_Inventory 41
9.0 Bundles 43
9.1 Generating and Populating a Bundle Label 43
9.2 Bundle Area 43
9.3 File_Area_Text 44
9.4 Bundle Member Entry 45
10.0 Deliveries 47
10.1 The Package 47
10.2 Manifest Files 47
10.2.1 The Checksum Manifest 47
10.2.2 The Transfer Manifest 48
10.3 Transfers Using the PDS4 Default Procedure 48
10.4 Generating and Populating an XML Label for the “Package” 49
PART V. LOCAL DICTIONARIES AND OTHER DOCUMENTATION 50
11.0 Local Data Dictionary 50
11.1 The Mission Area 51
11.2 The Discipline Area 52
11.3 Building and Using Local Data Dictionaries 52
12.0 Other documentation 61
12.1 Internal versus External Documentation 62

12.2 Restricted Documents 63
PART VI. VALIDATE THE ARCHIVE 64
13.0 Validation 64
13.1 Data Product Validation 64
13.2 PDS4 Validation Tool 65
APPENDIX A ACRONYMS AND ABBREVIATIONS 66
APPENDIX B SCHEMATRON REFERENCES 67
APPENDIX C XML CATALOG FILES 68
APPENDIX D: CREATING A LABEL TEMPLATE 70
APPENDIX E FIELD AND GROUP COUNTING IN NESTED STRUCTURES 75
APPENDIX F LID FORMATION RULES FOR PDS4 CONTEXT PRODUCTS 77
APPENDIX G PROCESS FOR SELECTING A PRODUCT CLASS 81
APPENDIX H PERMITTED SCHEMA MODIFICATIONS 84

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 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 that this version of the document is conformant to version “1.4.0.n” of the Information Model (IM).

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)[1]. 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[2] describing the PDS4 system.

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

[2] XML schema documents (XSD), Schematron files (SCH), the Information Model (IM) specification, and the Standards Reference are "views" into the IM and are collectively the "Standards".

Readers, even those very familiar with previous versions of PDS, **should read the *PDS4 Concepts* document [4] before beginning the *Data Providers' Handbook.***

The *DPH* should be used in conjunction with the *PDS Standards Reference* [2], the PDS4_PDS schema and Schematron files, and the *PDS4 Data Dictionary* [3], which collectively provide the information necessary to develop a PDS4 compliant archive.

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. Consult external sources for information on XML.

The use of an XML editor simplifies construction and validation of schemas (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 Applicable Documents

1.5.1 Controlling Documents

[1] *Planetary Data System (PDS) PDS4 Information Model Specification*, version 1.4.0.0

[2] *Planetary Data System Standards Reference*, v.1.4.0, September 2015.

[3] *PDS4 Data Dictionary*, version 1.4.0.0.

[4] *PDS4 Concepts*, version 1.4.0, August 2015.

1.5.2 Document Availability

PDS4 documents governing archive preparation are available online:

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

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

1.6 Other Resources

1.6.1 XML Schema location

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

1.6.2 PDS4 Software

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

1.6.3 PDS4 Wiki

The PDS4 Wiki is a primer for 'How to Create a PDS4 Label' and can be found at:

https://oodt.jpl.nasa.gov/wiki/display/pdscollaboration/How+to+Create+a+PDS4+Label+-+FOR+REVIEW

1.6.4 SBN PDS4 Wiki

This site contains information developed during a PDS4 prototype migration effort at the Small Bodies Node. It has been adapted for PDS4 data developers both inside and outside of the SBN.

http://sbndev.astro.umd.edu/wiki/SBN_PDS4_Wiki

PART I. PRELIMINARIES

2.0 BUILDING BLOCKS

2.1 Comments from past missions regarding archive development

These comments come from missions which have recently archived with PDS – lessons learned, things they wish they had done, or things they did do that improved their archiving experience.

- Think of archiving as an integral part of the mission, not an additional activity.

- Initiate contact between PDS and the mission early; face-to-face discussion is particularly useful.

- Design an archive schedule with mission activities in mind. (Setting SIS due dates within a week of the target date for integrating instruments with the spacecraft is not a good idea!)

- Identify archive conventions which the mission should establish early. This is as much about coordinating among groups within the mission as it is about coordinating the mission with PDS:
  - Determine what data products will be archived at each processing level (raw, calibrated, high level)
  - Establish the archive organization (define bundles and collections and how products will be assigned to them)
  - Agree on a versioning scheme for the components of the archive (what represents a new version of a data product or a document, and how will the versions be represented?)
  - Establish naming conventions for bundles, collections, products, directories, and files.
  - Agree on a set of common time formats (for example, decide whether to use YYYY-MM-DD or YYYY-DDD for dates)
  - Agree on a set of common data formats (for example, can most tabular data be archived in fixed-width ASCII tables?)

2.2 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.*, if a 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 *logical 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 into 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.

A few words have meanings that 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 'attribute' (for PDS) and 'XML attribute' to establish the context.

Warning: We have tried to avoid using terms that have strong PDS3 connotations when the PDS4 meanings are different. Unfortunately, the English language does not provide a sufficient set of meaningful, unique, unambiguous terms to meet all of our needs. Please do not infer meanings from past experiences – review carefully the PDS4 definitions.in the *PDS4 Glossary*, Appendix A in the *PDS4 Concepts* document [4].

2.3 Data

Four basic structural data formats are allowed in PDS4.

1. Homogeneous array structure
   - Suitable for images, spectra, spectral cubes, maps, etc.
   - The elements of an array are homogeneous (each has the same binary storage format)
   - The individual elements of any array are stored with their bytes in the order dictated by their scalar type.
   - PDS requires a specific storage order for array elements; see *Standards Reference* [2], Section 4A.

2. Repeating record structure
   - Suitable for tabular data in fixed-length records
   - May be either binary or character, but a single object must be defined as one or the other (not mixed).
   - The fields of a record may be heterogeneous.
   - Any single field must be homogeneous from one record to another; but formats may vary among fields within a single record.

The majority of PDS4 data can be supported by these two structures. For those PDS4 objects which cannot be supported by the above, there are two additional structures 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, and 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); no decoding software (e.g., Adobe Acrobat©) is required.
    - The attribute parsing_standard_id is used to identify the parsing standard to be used.

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

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
• homogeneous array structure	↔	Array_Base
• repeating record structure	↔	Table_Base
• parsable structure	↔	Parsable_Byte_Stream
• encoded structure	↔	Encoded_Byte_Stream

Here are a few rules (for a full description, see the *Standards Reference* [2], Section 4):

- Each digital object must be stored in one of the four basic data formats.
- A digital object must be contained in a single file (i.e., a digital object cannot span multiple files)

- A file may contain multiple digital objects.[3]
- Digital objects within a file are not required to use the same storage structure.
- When multiple digital objects are contained in a single file, each must be contiguous (they may not overlap in storage)

[3] Except for documents, where there is a limit of one object per file.

3.0 SCHEMA AND LABELS – AN OVERVIEW

Labels are fundamental to the PDS and its holdings; they describe both the content and format of products. They also allow links to be established among products — so that observational data can reference descriptions of the instrumentation which collected the bits, the spacecraft which hosted the instruments, and the organizations that supported the activity. They also help to bind related products into collections and related collections into bundles.

Labels are XML documents, and their development begins with XML schemas. PDS maintains a common-schema that serves as a library of generic definitions for each PDS4 product type; there can also be schemas for specialized disciplines (for example, geometry and cartography) and for specific missions.

In order for any XML document (including a PDS label) to meet the XML standard, it must be both "well formed" and "valid". A well-formed XML document must have correct XML syntax; a valid XML document must conform to the rules of an XML schema document (XSD) and, if applicable, an XML Schematron (SCH) document. Under PDS4, schemas provide the rules governing the structure and some of the content of each XML class, and Schematron documents further constrain the contents of those classes and attributes within classes.

If you prepare data for delivery to PDS, you will certainly be involved in creating labels; you may also be involved in creating discipline- or mission-specific local data dictionaries (LDDs). We will discuss the steps of archive creation in more detail in later sections; here we will provide an overview of the labeling process since it is fundamental to almost everything else in PDS archiving.

3.1 Archive Design

In consultation with your PDS discipline node (DN), review the products you expect to include in your archive. Group related products into collections and related collections into bundles. Depending on the size and number of products you may have more or fewer collections and more or fewer bundles. A very simple archive could fit into a single bundle with half a dozen collections, each holding a handful of products; in such a case, you might want to collaborate with others and jointly produce one bundle, where your data would be in one of several contributed collections. A large archive could have many bundles and hundreds of data collections.

3.2 Pipeline Considerations

If you are developing an archive 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 a label template (xml) file as input to the pipeline software, or
b) Use a schema (xsd file) as input to the pipeline software you use to generate your labels.

A feature of many XML editors is the ability to generate a label template. A label 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 the software is specifically designed to handle XML.

In this *Handbook*, we assume the pipeline will use an XML template[4]. This provides a framework for the discussion that follows. Our goal here is to become familiar with the content of an archive and to illustrate the process by which it is created; we do not dwell on subtleties or exhaustively explore options.

3.3 Label Design

For each product type, determine which product labels you will need. Your consulting DN may assist you in developing individual labels by producing template XML files from the common-schema, possibly incorporating both discipline and mission specific areas if appropriate for the products you plan to produce. Section 7 in this document describes how to edit label templates after they have been created using an XML aware editor (Appendix D).

PDS supports two software initiatives for developing label design tools; these are more 'interactive' than the template editing process — the software asks you questions and then builds the label as you go along. The PDS User Centered Design (UCD) team at NASA's Ames Research Center has released its LACE tool, and a group at JPL has released the Label Design Tool (LDT) for its AMMOS-PDS Pipeline Service (APPS). Both of these are development projects; but the tools can be accessed through password-protected web sites. Your consulting DN can put you in touch if you are interested.

PDS also has a Generate Tool which provides a command-line interface for generating PDS4 Labels from either a PDS3 Label or a PDS3-specific Document Object Model[5] (DOM) object. This could help if you are migrating PDS3 data rather than building a new PDS4 archive from scratch. The Generate Tool allows you to generate PDS4 labels having values that may or may not map directly to the input data object (i.e. the keywords and keyword values in the original PDS3 label). For more information, go to:

http://pds.nasa.gov/pds4/software/generate/

[4] Option (b) is discussed briefly in Appendix H.
[5] The Document Object Model (DOM) is a programming interface for HTML and XML documents. It provides a structured representation of the document and it defines a way that the structure can be accessed from programs so that they can change the document structure, style and content.

If the existing schemas are not adequate to describe your data, you may need to build a Local Data Dictionary (LDD), either for a new discipline or for your mission or investigation. LDDTool is a good way to build a local dictionary; but you need an input file in order to run it.

For more detailed information, see Section 11.

3.4 Label Validation

Finished labels must be validated against the common-schema, the common-Schematron file, and any discipline and mission schema and Schematron files on which they are based. Thus, in order for a label to be compliant with PDS4 standards, the label must:

- Have correct XML syntax
- Be compliant with the class and attribute structures defined by the PDS common-schema and any relevant discipline and mission dictionary schemas (XSDs).
- Be compliant with the rules governing specific attributes and their values as set by the PDS Schematron files and any relevant discipline and mission Schematron files.

Some XML aware editors provide real-time validation of labels once you have loaded them into your computer. This feature is particularly handy when you are editing templates since you can see where the errors lie and correct them one at a time.

PDS also has a Validate Tool which checks completed labels against schemas and Schematron files. For more information, go to:

https://pds.nasa.gov/pds4/software/validate/

3.5 Other Resources

There are a number of other sources of information and tools for label design and generation. The Small Bodies Node PDS4 Wiki is one (see section 1.6.4 of this document).

PART II. THE FIRST STEPS

Assuming you have an instrument with known data products, the first step in designing an archive is to organize the data products into collections and the collections into bundles.

4.0 OUTLINE THE BUNDLE

For simplicity, imagine that your archive includes a single table of observational data (and its label) and that the remainder of the archive consists of supplementary information, which will help future scientists understand and use the data. We will organize this material into a single bundle with five collections:

Browse Collection
Context Collection
Data Collection
Document Collection
XML Schema Collection

The data, obviously, will be in the data collection; an abbreviated or reduced resolution version of the data will be in the browse collection. Background information on the mission, instrument, observations, etc. will be in the context collection, while actual documents will go to the document collection. The XML schema collection contains common, discipline, and mission schemas and Schematron files used in constructing and validating product labels.

4.1 Directory Organization

PDS has provided a default procedure for organizing data that will be transferred to or from (or moved within) the PDS (see *Standards Reference* [2], Section 2B.2). Your bundle can be organized into a simple directory structure with one directory in the bundle root for each collection, as shown below. Note that we have simplified the structure by showing products — rather than data file/label pairs in several cases.

```
bundle root
| - bundle.xml
| - readme.html
|
| - browse
|    | - collection_browse.xml
|    | - collection_browse_inventory.csv
|    |
|    | - browse_ product
|
| - context
|    | - collection_context.xml
|    | - collection_context_inventory.csv
|    |
|    | - context_product1
|    | - context_product2
|    | - context_product3
|
| - data
|    | - collection_data.xml
|    | - collection_data_inventory.csv
|    |
|    | - data_file.tab
||   | - data_file.xml
|
| - document
|    | - collection_document.xml
|    | - collection_document_inventory.csv
|    |
|    | - errata_document_product
|    | - mission_document_product
|    | - instrument_document_product
|
| - xml_schema
     | - collection_xml_schema.xml
     | - collection_xml_schema_inventory.csv
     |
     | - xml_schema_product1
     | - xml_schema_product2
     | - xml_schema_product3
```

The root level subdirectories each correspond to a single collection. Each directory contains a collection Inventory file and its XML label file.

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

4.2 Directory and File Naming

In the example above, there is a single observational data file in the data directory, data_file.tab. Had there been a large number of observational data files, it would have been reasonable for you to establish subdirectories under the data directory — for example, having directory names based on the UTC date, each directory containing files with names based on the UTC time. There are many other ways to name directories and files. As was discussed in Section 2.1 of this document, it is often advantageous to negotiate these naming conventions well in advance, in the context of the entire mission, and with regard to naming conventions adopted by past missions with similar instruments. Using the same base name for label and data file names is considered good practice; but this is not a PDS4 requirement and is not always possible, such as when a product consists of two or more data files.

4.3 Determine the Documentation Needed

Although it is not a structure issue, you and your consulting DN data engineer should sketch out the contents of the document collection very early in the design process. PDS requires that products, collections, and bundles be documented so that scientists in future years can understand (1) how the data were collected and processed, (2) what the data mean, and (3) the limitations of the data. You should refer to the *Standards Reference* [2], Section 8, and confer with your data engineer to determine all of the required and appropriate documentation for the document collection.

Documentation takes three forms in PDS: (a) XML labels, (b) documents included within the archive, and (c) references to material that is publicly available elsewhere. The archive design should include a clear plan for how the documentation will be distributed among these three categories.

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 the *PDS4 Information Model* [1] for a list of PDS approved formats.

As an example, documentation might include the following:

1. An errata file that describes any changes, known errors, or anomalies 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. While an MD5 checksum file is not required, it provides a means for future users to confirm the integrity of the files they download from PDS (except for the MD5 checksum file itself).

Each of the above document products is individually labeled. Both the errata and checksum files array be labeled using the Product_File_Text class definition, as these documents are strictly ASCII text. The other two document products, the mission and instrument descriptions, are labeled using the Product_Document class definition 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).

5.0 DESIGN LOGICAL AND VERSION IDENTIFIERS

Every product has an identifier that is unique across all products registered and 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.

5.1 General Concepts

Here are some general rules:

- LIDs must be unique across PDS
- One LID covers all versions of a single product; the VID distinguishes among versions[6]
- Each PDS4 LID is constructed as a Uniform Resource Name (URN)
- Each LID in a PDS archive begins with an agency identifier (namely, 'urn:nasa:pds:')[7]
    - This ensures that any LID which is unique within PDS is also unique within NASA and within the global federation of agencies which subscribe to this identification system — that is, the prefix ensures that products have globally unique identifiers.
- LIDs are restricted to lower-case letters, digits, the dash, the period, and the underscore. Colons are also used but only in a prescribed way to delimit "fields" (see below)
- Each PDS4 LID is constructed as four (bundle), five (collection), or six (product) fields, where each pair of fields is separated by a colon. Each field must begin with a letter or digit.
- LID maximum length is 255 characters.

5.2 Constructing LIDs

Detailed requirements and formation rules for LIDs are provided in the *Standards Reference* [2], Section 6D.2; we provide a brief summary here.

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

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

---

[6] Discuss with your consulting DN the conditions under which a new version should actually be a different product. Versions generally succeed each other and result from (minor) improvements — incrementally better calibration, for example. If the processing algorithm itself changes significantly, you may want to consider defining a new set of products rather than incrementing the version.

[7] You may find other prefixes in archives maintained by other agencies (for example, urn:esa:psa: for the PSA archive), but do not use these when preparing data for delivery to PDS.

- Bundle LIDs -- are constructed by appending a unique bundle specific field to the agency identifier -- ‘urn:nasa:pds’ or ‘urn:esa:psa’.

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

  Since all PDS bundle LIDs are constructed this way, the bundle LID will be globally unique.

- Collection LIDs -- are constructed by appending a unique collection field to the parent bundle’s LID, for example

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

  Since all PDS collection LIDs are constructed in this way and the collection field is unique within the bundle LID, the collection LID will be globally unique.

- Basic Product LIDs -- are constructed by appending a unique 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 product LID will be globally unique.

5.2.1 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 instrument team decides to use the spacecraft clock count (sclock) 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

In the above example bundle field, ssc01.hirespc.cruise, we used periods as separators. Alternatively, we could have used dashes, underscores, or some combination of the three. Discuss the use of period, dash, and underscore in LIDs with your consulting node; there may be a preference.

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:xml_schema

If there had been a large number of products, it might have been desireable to subdivide the data products into two or more collections by processing level (e.g., data_raw, data_derived, etc.), by year (data_2006, data_2007, etc.), or by a different discriminator. Many discriminators are permitted; you should use whichever is best suited to your data. Again, early discussions with your consulting node are strongly encouraged.

Products [examples of data products in various collections 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

5.3 VID Construction

Detailed requirements and formation rules for version identifiers (VIDs) are provided in the *Standards Reference* [2], Section 6D.3; we provide a brief summary here.

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

- VIDs are appended to LIDs by a double colon ("::").
- VIDs must be of the form `M.n.o.p` where `M` is the most significant field and subsequent fields have decreasing significance in left-to-right order, "`M`", "`n`", "`o`" and "`p`" are integers. The "`o`" and "`p`" fields are not used in some applications, such as for observational data.
- The major number (`M`) is initialized to '1' for archive products, but the number '0' may be used for sample products or test run products that are not yet ready for the archive. Whenever the major number (`M`) is incremented, the minor number (`n`) is reset to '0'.
- Neither `M` nor `n` should be prepended with zeros; each is simply incremented as an integer. Thus "1.1" and "1.10" are different versions, and "1.01" is invalid.

5.4 LIDVID Construction

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

The following example LIDVIDs are based on the example LIDs in Section 5.2.1. In all cases the VID is "1.0".

5.4.1 Examples

Note that in the example LIDs that follow, we used periods as separators. Alternatively we could have used dashes, underscores, or some combination of the three. Discuss the use of period, dash, and underscore in LIDs with your consulting node to determine if the node has a preference.

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:xml_schema::1.0

Products [examples of 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

5.5 LIDVIDs – The Next Step

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 the pipeline that automatically generates the labels in your archive.

LIDs and LIDVIDs will be ubiquitous in your archive. Be sure you have the naming convention, formation rule, and/or algorithm correct before proceeding. When you have constructed draft LIDs and LIDVIDs, contact your PDS DN to verify they are unique and are conformant.

PART III. BASIC PRODUCT LABELS

6.0 BASIC PRODUCT LABELS - AN OVERVIEW

We can think of the material in the PDS archive as either observational or supplementary data (see Appendix A in the *PDS4 Concepts* document for formal definitions). The dividing line is not strict; for example, observational data from one investigation may be calibration (supplementary) data for another. Data providers should work with consulting discipline nodes to determine the classification that makes most sense both within an archive and across PDS following the steps outlined in Section 6.1 below.

All products in PDS require labels. We discuss labels for observational products first; then we discuss aspects of non-observational (supplementary) product labels, which differ slightly. Aggregate product labels and their construction are discussed in Sections 8 (collections) and Section 9 (bundles).

You and a data engineer from the consulting DN should discuss the nature of the files that will be included in your archive, roughly categorizing the products according to whether they are observational, supplementary, or aggregate. Once the data engineer has sufficient information, the appropriate products (e.g., Product_Observational, Product_Document, Product_Collection, etc.) and the associated controlling files — common, discipline, and mission schema and Schematron files — can be identified. This set of schemas and Schematron files defines the products in your archive, can be used to generate label templates, and provides validation criteria for ensuring the integrity of the products in your archive. If you don't have them already, the data engineer can provide the files to you.

All product labels are XML files generated and validated against a specific version of the common schema and applicable LDD schemas maintained by the PDS.

This set of schemas (e.g., common, discipline, and mission schemas) and Schematron files defines the products in your archive and provides validation criteria for ensuring the integrity of the products in your archive.

6.1 Product Selection

Each PDS4 label must identify the type of product the label is describing. As discussed later, the various "Product" classes provide the set of options from which you choose the XML label's Root Tag. The options include:

- **Product Browse** - a basic product containing a low resolution or "quick-look" version of an observational product.

- **Product Bundle** - an aggregate product used to identify the member collections of an archive bundle.
- **Product Collection** - an aggregate product used to identify the member basic products of an archive collection.
- **Product Context** - a basic product identifying the physical (instrument, spacecraft, target, people) and conceptual (investigation, node) objects related to an observational product's provenance.
- **Product Document** - a basic product identifying a single logical document, such as an interface specification, instrument description, or user's manual; the document product may be archived in multiple formats under the single logical Product_Document definition.
- **Product File Text** - a basic product consisting of a single digital file with ASCII character encoding.
- **Product Observational** - a basic product comprising one or more images, tables, and/or other fundamental data structures that are the result of a science or engineering observation.
- **Product SPICE Kernel** - a basic product consisting of a SPICE kernel.
- **Product Thumbnail** - a basic product consisting of a highly reduced version of an image, typically used in displaying the results from search interfaces.
- **Product Update** - a basic product containing information about updates to observational products that have already been archived.
- **Product XML Schema** - a product consisting of XML formatted schemas, Schematron files, OASIS catalog files, or any other reference schemas used in the interpretation of an observational product

Note that the class, Ingest_DD, used to generate a local data dictionary (see Section 11) is also a permitted option for Root Tag. It is the only class other than the Product_* classes which may be used for Root Tag.

The selection of appropriate product classes, when the choice is not obvious, is outlined in Appendix G – Process for Selecting a Product Class

6.2 Basic Product Labels – Observational Products

6.2.1 Selecting the Structural Description for Observational Products

Based on the nature of the observational data, you and your PDS consulting node data engineer will determine the classes to use for describing the observational objects (e.g., Array, Table, Header, etc.) in your archive. You may have already determined these if you followed the procedure outlined in Section 6.1. Product_Observational allows the following:

For binary array data —

- Array
- Array_2D
- Array_2D_Image

- Array_2D_Map
- Array_2D_Spectrum
- Array_3D
- Array_3D_Image
- Array_3D_Movie
- Array_3D_Spectrum

For binary tabular data —
- Table_Binary

For character tabular data —
- Table_Character (fixed width fields)
- Table_Delimited (variable width fields)

The following are allowed, but are less frequently used; special constraints may apply:
- Header
- Encoded_Header
- Encoded_Binary (Supplemental)
- Encoded_Byte_Stream (Supplemental)
- Encoded_Image (Supplemental)
- Parsable_Byte_Stream (Supplemental)
- Stream_Text

Any basic product may include multiple objects and multiple object types. For example, you may have an Array_2D_Image, a Table_Character histogram of pixel values, and a Header. The three objects, which may or may not all reside in the same file, coupled with the single XML label (that describes these objects), form the single digital product. The File_Area_Observational class permits inclusion of multiple digital objects (e.g., Header, Table_Character, etc.) that are in a single file. If there are several files, then a separate File_Area_Observational must be used for each file.

If your archive has browse objects, you may either create separate browse products and associate them with the corresponding observational products using the Reference_List class, or incorporate them into the observational products using File_Area_Observational_Supplemental. For additional information and advice, contact your consulting DN.

6.2.2 Basic Product Label Organization – Observational Products

In a typical Product_Observational label there are several major blocks of information (areas). Each 'area' contains one or more classes, each of which may have several attributes. The areas, plus some XML overhead at the beginning, are outlined below:

- XML Prolog
    - Provides a declaration that the label is an XML versioned document
    - Identifies the schema and Schematron files against which the label is validated.

- Root Tag
    - Provides a declaration of the root XML element of the label. The Root Tag is based on the product type — for example, Product_Observational.
    - Identifies the namespaces used in the label (i.e., the 'pds' namespace plus any additional discipline or mission namespaces) including the associated schema.

- Identification Area
    - Provides identification information specific to the product (the LID and VID, the information model version, the product class, etc.)
    - Provides citation information specific to the product
    - Provides modification history specific to the product

- Observation Area
    - Provides information about the investigation, instrument, target, times, etc.
    - Includes subsections for relevant classes specific to one or more discipline nodes, and to the mission (or an equivalent namespace).

- Reference List Area
    - Provides identification information for products, journal articles, etc., relevant to understanding the product. References may be made to sources both internal and external to PDS.

- File Area
    - Identifies the file(s) containing the digital object(s), and
    - Provides classes specific to each digital object in a given file (e.g., the description and parameters of each header, table, image).

The example in Figure 6-1 shows the major blocks of information (areas) in a typical basic product label.

With two exceptions, all Product_Observational labels have the structure shown in Figure 6-1. The exceptions are:

- Reference_List is optional; and

- One or more File_Area_Observational_Supplemental classes, which allow supplementary data objects to be carried along with observational data products, may be added.

This structural consistency simplifies extraction of information from such labels, facilitating search and retrieval of products by future users. Note, however, that there is considerable flexibility for content within the areas shown in Figure 6-1.

XML Prolog

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-model href="https://pds.nasa.gov/pds4/pds/v1/PDS4_PDS_1400.sch"
            schematypens="http://purl.oclc.org/dsdl/schematron"?>
```

Root Tag

```
<Product_Observational xmlns="http://pds.nasa.gov/pds4/pds/v1"
  xmlns:pds="http://pds.nasa.gov/pds4/pds/v1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://pds.nasa.gov/pds4/pds/v1
      https://pds.nasa.gov/pds4/pds/v1/PDS4_PDS_1400.xsd">
```

Areas

```
<Identification_Area>
     27 lines of detail
</Identification_Area>

<Observation_Area>
     76 lines of detail
</Observation_Area>

<Reference_List>
     10 lines of detail
</Reference_List>

<File_Area_Observational>
     41 lines of detail
</File_Area_observational>
```

End Tag

```
</Product_Observational>
```

Figure 6-1. Example label structure for Product_Observational

***For additional information regarding the definitions and cardinality of the attributes and classes used within the definition of a basic product label, consult the PDS4 Data Dictionary* [3].**

.

6.3 Basic Product Labels – Non-Observational Products

Labels for non-observational products are structurally similar to those for observational products; but there are more product types, and those product types have more variations. This section highlights the differences.

6.3.1 Selecting the Appropriate Types of Non-Observational Products

Common product types for supplementary data include:

- Product_Ancillary
- Product_Browse
- Product_Context
- Product_Document
- Product_File_Text
- Product_SPICE_Kernel
- Product_Thumbnail
- Product_XML_Schema

Unlike Product_Observational, which supports multiple object structures, some of the non-observational product types support only one structure. For example, Product_Thumbnail may only describe an encoded image and only seven encoding standards are recognized (GIF, JPEG2000, JPEG, PDF, PDF/A, PNG, and TIFF).

6.3.2 Basic Product Label Organization – Non-Observational Products

The label for a supplementary product is similar to the label for Product_Observational; but the many options for the different product types make a general description impossible. Supplementary product labels begin with the XML Prolog and Root Tag, and they end with the End Tag (see Figure 6-1). They include the Identification_Area and (optionally) the Reference_List; but, after that, the customization for each supplementary product type defies generalization. Check the *PDS4 Information Model* [1] and work closely with your consulting DN when designing these labels.

6.4 Aggregate Product Labels

There are two types of aggregate products:

- Product_Collection
- Product_Bundle

These product types are discussed in detail in Sections 8 and 9, respectively.

7.0 LABEL TEMPLATE CREATION AND EDITING

An XML label template is a file that can be used to generate labels for a single product type in your archive. It looks like a final label, but it has dummy variables or placeholders where the real values will be substituted by your pipeline software (or by you, manually). Some templates have no values rather than placeholders.

You can use the functions provided by an XML aware editor (XAE) to generate label templates from the common-schema (xsd). You (or your consulting DN data engineer) will need to generate a label template for each type of product that will appear in your archive (bundle, collection, and each type of basic product). If you have several different product types with the same structure — for example, different ASCII tables for solar flux and magnetic field — you will probably need separate templates for each.

Detailed instructions for generating templates using an XAE are given in Appendix D. The remainder of this section walks you through the process of editing a template after it has been generated so that it can be used repeatedly with your pipeline software. We will also mention Schematron 'rules' which can be formulated to enforce certain conditions.

7.1 Label Template Editing — Body

Although the XML Prolog and Root Tag appear at the begining of the label template, we will discuss editing of the 'body' of the template first. We will return to the XML Prolog and Root Tag later.

The modifications that you can make to the body of the label template are limited. Keep in mind that all changes to the label template must adhere to the constraints dictated by the referenced versions of common, discipline, and mission schemas and Schematron files. You can:

1. remove classes or attributes designated as optional in the parent schema (e.g., the "common-schema" and any locally-defined schema).
2. ensure that an optional class and/or attribute is present in the label template
3. modify the number of times a class or attribute is repeated
4. set an attribute to a fixed value, set it to a value from an enumerated list, or identify it as variable

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

Examples in the following sections show fragments of XML templates. Values that vary from one label to the next are represented by placeholders, which either you or the pipeline software will replace when you generate real labels. These typically contain dummy values — for example, the value "name1" may appear in the XML element "<name>name1</name>"; you or your pipeline software will search the template for "name1" and substitute the actual name.

7.1.1 Presence or Absence of a Class or Attribute

In your label template you can remove, add, or replicate classes and attributes so long as you do not stray outside the limits in the controlling schema(s). Use your favorite text editor or XAE to make the edits. Note that using an XAE allows you to validate in real time as you make your edits.

In the label template fragment below:

```
<Field_Character>
    <name>name1</name>
    <field_number>field_number1</field_number>
    <field_location unit="unit1a">field_location1</field_location>
    <data_type>data_type1</data_type>
    <field_length unit="unit1b">field_length1</field_length>
    <field_format>field_format1</field_format>
    <unit>unit1c</unit>
    <scaling_factor>scaling_factor1</scaling_factor>
    <value_offset>value_offset1</value_offset>
    <description>description1</description>
    <Special_Constants>
         special constants detail
    </Special_Constants>
    <Field_Statistics>
         field statistics detail
    </Field_Statistics>
</Field_Character>
```

removing all of the optional attributes and optional classes leaves you with

```
<Field_Character>
    <name>name1</name>
    <field_location unit="unit1a">field_location1</field_location>
    <data_type>data_type1</data_type>
    <field_length unit="unit1b">field_length1</field_length>
</Field_Character>
```

7.1.2 Ensuring the Presence of a Class or Attribute

There may be circumstances in which you want to ensure that an optional class or attribute is included in the label. You can do this by adding a ‘rule’ to a Schematron file. For example, the optional attribute “description” was removed in the editing example above. If you wanted to ensure that “description” was always included in “Field_Character”, you could insert a constraint, such as the one below, that would generate an error message if “description” were missing. Note that the constraint would have to be added to a discipline or mission Schematron file, since you may not modify the common-schema and Schematron files.

```
<sch:pattern>
   <sch:rule context="//pds:Field_Character">
      <sch:assert test="pds:description">
          The description in Field_Character must exist.
```

```
            </sch:assert>
        </sch:rule>
    </sch:pattern>
```

See Appendix B for more information on how to construct Schematron 'rules', or consult your discipline node.

7.1.3 Repeating a Class or Attribute

If you need more than one instance of a class or attribute in your label template, you can replicate the necessary structure. For example, if a character record has two fields, you can replicate the Field structure shown in Section 7.1.1 (we have omitted the optional attributes "scaling_factor" and "value_offset" and the optional classes "Special_Constants" and "Field_Statistics" from the "Field" definitions). The repeated structure is shown in red below.

```
    <Record_Character>
        <fields>fields1</fields>
        <groups>groups1</groups>
        <record_length unit="unit1a">record_length1</record_length>
        <Field_Character>
            <name>name1</name>
            <field_number>field_number1</field_number>
            <field_location unit="unit1b">field_location1</field_location>
            <data_type>data_type1</data_type>
            <field_length unit="unit1c">field_length1</field_length>
            <field_format>field_format1</field_format>
            <unit>unit1d</unit>
            <description>description1</description>
        </Field_Character>
        <Field_Character>
            <name>name2</name>
            <field_number>field_number2</field_number>
            <field_location unit="unit2b">field_location2</field_location>
            <data_type>data_type2</data_type>
            <field_length unit="unit2c">field_length2</field_length>
            <field_format>field_format2</field_format>
            <unit>unit2d</unit>
            <description>description2</description>
        </Field_Character>
    </Record_Character>
```

Here is an example of a Schematron 'rule' that ensures that exactly 5 fields are defined in the specification of a character record:

```
<sch:pattern>
   <sch:rule context="pds:Record_Character">
      <sch:assert test="count(pds:Field_Character) eq 5">
         There must be 5 and only 5 instances of Field Character
      </sch:assert>
   </sch:rule>
</sch:pattern>
```

7.1.4 Setting the Value of an Attribute (Fixed and Variable)

For attributes with fixed values — like the number of fields in a record of a fixed width table — you can insert the appropriate value directly into the template. For attributes that will vary from one table to another, you should use a placeholder, which can be identified and replaced by your pipeline processing software. In the example shown in Section 7.1.3, assume that the data files list temperature as a function of time but that some tables use Celsius and others use Kelvin. The label template can be fleshed out as follows (red denotes values that you have inserted):

```
<Record_Character>
    <fields>2</fields>
    <groups>0</groups>
    <record_length unit="byte">22</record_length>
    <Field_Character>
        <name>Time</name>
        <field_number>1</field_number>
        <field_location unit="byte">1</field_location>
        <data_type>ASCII_Real</data_type>
        <field_length unit="byte">9</field_length>
        <field_format>f9.2</field_format>
        <unit>second</unit>
        <description>Time in seconds past midnight</description>
    </Field_Character>
    <Field_Character>
        <name>Temperature</name>
        <field_number>2</field_number>
        <field_location unit="byte">11</field_location>
        <data_type>ASCII_Real</data_type>
        <field_length unit="byte">9</field_length>
        <field_format>f9.2</field_format>
        <unit>unit2d</unit>
        <description>description2</description>
    </Field_Character>
</Record_Character>
```

All of the attribute values are constant except for “unit” and “description” in field 2. The placeholders “unit2d” and “description2” will be replaced by the pipeline processing software depending on whether the table contains values in “degC” or “K”.

To ensure that an attribute appears in your XML label with the correct value you can add a ‘rule’ to your discipline or mission XML Schematron file. The Schematron ‘rule’ below reports an error if “information_model_version” in class “Identification_Area” is not set to “1.0.0.0”.

```
<sch:pattern>
  <sch:rule context="pds:Identification_Area">
  <!-- ============================================================ -->
  <!-- Test: Does 'information_model_version' match expected value?   -->
  <!-- ============================================================ -->
  <sch:assert test="pds:information_model_version='1.0.0.0'">
    Identification_Area.information_model_version: does NOT specify
    the correct version. </sch:assert>
  </sch:rule>
</sch:pattern>
```

To ensure that the value of an attribute matches one of the values from an enumerated list of values, use a ‘rule’ like the following. This Schematron fragment will generate an error message if attribute “encoding_type” in class “SPICE_Kernel” is neither “Binary” nor “Character”.

```
  <sch:pattern>
    <sch:rule context="pds:SPICE_Kernel">
      <sch:assert test="pds:encoding_type = ('Binary', 'Character')">
        The attribute pds:encoding_type must be equal to one of the
        following values 'Binary', 'Character'.</sch:assert>
    </sch:rule>
  </sch:pattern>
```

See Appendix B for more information on how to construct Schematron ‘rules’, or consult your discipline node.

7.2 Label Template Editing — XML Prolog and Root Tag

This section addresses how the XML Prolog and Root Tag of an XML label are created; both are always at the beginning of a PDS4 XML label. Angle brackets — “<” and “>” — delimit XML ‘tags’, which may extend over several lines. In the example below the first three XML tags comprise an example XML Prolog; the fourth is an example Root Tag. Note, however, that each XAE has different requirements; consult your XAE user guide or your consulting DN for more information.

```
<?xml version="1.0" encoding="UTF-8">
<?xml-model href="https://pds/nasa/gov/pds4/pds/v1/PDS4_PDS_1400.sch"
 schematypens="http://purl.oclc.org/dsdl/schematron"?>

<?xml-model href="https://pds.nasa.gov/pds4/img/v1/PDS4_IMG_1400.sch"
 schematypens="http://purl.oclc.org/dsdl/schematron"?>

<Product_Observational
  xmlns="http://pds.nasa.gov/pds4/pds/v1"
  xmlns:pds="http://pds.nasa.gov/pds4/pds/v1"
  xmlns:img="http://pds.nasa.gov/pds4/img/v1"
  xmlns:mpf="http://pds.nasa.gov/pds4/mission/mpf/v1"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="
           http://pds.nasa.gov/pds4/pds/v1
           https://pds.nasa.gov/pds4/pds/v1/PDS4PDS_1400.xsd
           http://pds.nasa.gov/pds4/img/v1
           https://pds.nasa.gov/pds4/schema/released/PDS4_IMG_1400.xsd
           http://pds.nasa.gov/pds4/mission/mpf/v1
           https://pds.nasa.gov/pds4/schema/released/PDS4_MPF_1400.xsd">
```

7.2.1 XML Declaration Statement

The first XML tag in your label template declares the version and encoding of the XML document; it is created automatically from the common-schema (XSD), if you use an XAE to generate your label template. Typically, it looks like this:

```
<?xml version="1.0" encoding="UTF-8">
```

If the XML document has only ASCII characters, then the following may be substituted:

```
<?xml version="1.0" encoding="ASCII">
```

More information about the XML Declaration Statement can be found at:

http://www.w3.org/TR/REC-xml/#sec-rmd

7.2.2 Schematron References

The second XML tag is a declaration of how to locate the common-Schematron validation file; it must be entered into your label template manually.

```
<?xml-model href="https://pds.nasa.gov/pds4/pds/v1/PDS4_PDS_1400.sch"
 schematypens="http://purl.oclc.org/dsdl/schematron"?>
```

The `href` XML attribute points to the PDS4 common-Schematron file.

Additional discipline or node-specific Schematron files may be referenced using similar notation. These XML tags must be added immediately after the common-Schematron XML tag. For example, the third XML tag in Section 7.2 specifies a PDS Imaging Node camera Schematron file:

```
<?xml-model href="https://pds.nasa.gov/pds4/img/v1/PDS4_IMG_1400.sch"
 schematypens="http://purl.oclc.org/dsdl/schematron"?>
```

All currently available PDS4 schemas and Schematron files can be found at the following URL:

```
https://pds.nasa.gov/pds4/schema/released/
```

Information about the namespace for the Schematron validation language can be found at:

http://purl.oclc.org/dsdl/schematron

This URL is included as the value of the XML attribute `schematypens` in both of the example XML tags above.

Unfortunately, as suggested at the beginning of Section 7.2, there are complications. The `xml-model` XML tags tell your XAE where to find the relevant Schematron files. In the examples above we used URLs that point to the permanent PDS4 repository. While some XAEs (e.g., oXygen) support this approach, others do not. The current version of Eclipse was written before `xml-model` accommodated remote references, and Eclipse is limited to local references — that is, to XML tags like the following:

```
<?xml-model href="../../schema/pds/v1/PDS4_PDS_1400.sch"
schematypens="http://purl.oclc.org/dsdl/schematron"?>
```

While it is prefereable to avoid the use of explicit local file system directories in an archive product, users of the Eclipse XAE may have no other option if they want real-time validation.

7.2.3 Catalog Files

When building product labels, you might want to reference the permanent location of schemas and Schematron files once your archive is complete and registered with the PDS but still retain the option of using local files for development and when you are not connected to a network. It would be cumbersome to edit every label each time you switched back and forth. If your XAE supports them, an XML 'catalog' file may be used to map the permanent location listed in the label to a working location on your file system. See Appendix C for more information on XML catalog files.

7.2.4 Root Tag – Product Type and Namespace Specifications

The XML Declaration Statement and Schematron References are followed immediately (fourth XML tag in the example in Section 7.2) by the Root Tag, which is the highest (first) XML tag. It is auto-populated when you generate your template using an XAE.

There are two components to the Root Tag. The first specifies one of the approved PDS product types (e.g., Product_Document, Product_Collection, or Product_Observational). The second is a set of reference pairs between the URI of the component (which is derived from the schema that defines the PDS product) to an actual file containing the schema. The Root Tag looks like this:

```
<Product_Observational
  xmlns="http://pds.nasa.gov/pds4/pds/v1"
  xmlns:pds="http://pds.nasa.gov/pds4/pds/v1"
  xmlns:img="http://pds.nasa.gov/pds4/img/v1"
  xmlns:mpf="http://pds.nasa.gov/pds4/mission/mpf/v1"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation=
        http://pds.nasa.gov/pds4/pds/v1
        https://pds.nasa.gov/pds4/pds/v1/PDS4_PDS_1400.xsd
        http://pds.nasa.gov/pds4/img/v1
        https://pds.nasa.gov/pds4/schema/released/PDS4_IMG_1400.xsd
        http://pds.nasa.gov/pds4/mission/mpf/v1
        https://pds.nasa.gov/pds4/schema/released/PDS4_MPF_1400.xsd
```

Immediately following the product type (`Product_Observational` in this example), should be a list of namespaces that are used in the label. `xmlns` specifies the default namespace; it is the same as the `xmlns:pds` namespace, which is given on the following line. The `pds` namespace is where all PDS4 common classes and attributes are defined. All PDS4 labels should have the default namespace set as the `pds` namespace.

If there are discipline or mission local dictionaries used in the label, their definitions should follow; `xmlns:img` and `xmlns:mpf` specify Imaging and Mars Pathfinder dictionaries, respectively, in this example.

`xmlns:xsi` is an abbreviation for the XML Schema Instance namespace formally designated by the URL:

http://www.w3.org/2001/XMLSchema-instance

`schemaLocation` is an XML attribute of the Root Tag, which is defined in the `xsi` namespace; we say more about `schemaLocation` in the next section.

In an XML document with a default namespace, bare elements and attributes (i.e., those that are not prefixed with a namespace abbreviation followed by a colon) are assumed to belong to the default namespace.

7.2.5 Root Tag – Schema Location

The last XML attribute in the Root Tag example above specifies the location of the XML schemas that are referenced by the XML document. There are several ways to do this; we discuss two below.

The `xsi:schemaLocation` XML attribute contains pairs of values, separated by one or more spaces. The first value of a pair is a namespace; the second is the location of the XML schema for that namespace. In the example below, the `pds` namespace is shown as the first value, and the URL for the PDS4 common-schema, which defines all the PDS classes and attributes for that namespace, is listed second.

```
xsi:schemaLocation="
   http://pds.nasa.gov/pds4/pds/v1
   http://pds.jpl.nasa.gov/pds4/schema/released/pds/v1/PDS4_PDS_1400.xsd"
```

In the next example, the "`pds`" namespace is again shown as the first value; but the second value is the path to the physical location of the schema file on the local computer.

```
 xsi:schemaLocation="
   http://pds.nasa.gov/schema/pds4/pds/v1
   file:/D:/schemas/PDS4_PDS_1400.xsd"
```

When you create your label template using an XAE, the Root Tag is auto-populated. The `xsi:schemaLocation` namespace is derived from the schema, and the location is the path to the copy of the common-schema file on your local computer. However, you will most likely need to change the location as the label template and schema progress through the development stage. Clearly, final labels should not contain a location specific to the computer you used to generate the template; they should point to the permanent location of the XSD file within PDS. You can use that location from the outset by obtaining the correct URL from your consulting node, and using a catalog file to associate the local version of the file with the permanent location.

For additional information on using an XML catalog to reference the schema, see Appendix C on XML Catalog Reference.

7.3 XML Label – The End Tag

In order for the XML label to be well formed, it must end with a closing tag for the class opened at the beginning of the label.

```
<Product_Observational
  xmlns="http://pds.nasa.gov/pds4/pds/v1"
  xmlns:pds=http://pds.nasa.gov/pds4/pds/v1
           .
           .
           .
</Product_Observational>
```

7.4 Next Steps

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

For additional information regarding the XML Prolog and the Root Tag and how they are used within a Product Label, consult your discipline node.

Another resource for information on the XML Prolog is http://borrelly.astro.edu/wiki/Anatomy_of_the_XML_Prolog

PART IV. COLLECTIONS, BUNDLES, DELIVERY PACKAGES

8.0 COLLECTIONS

The next higher level in the organizational hierarchy of an archive is the collection — an inventory of member products and an accompanying label. The inventory and label are known as a collection product. Basic products of similar type and content are grouped into a collection. Observational data, for example, will be gathered into one or more data collection(s), documents into a document collection, etc.

With your consulting DN, determine how best to group your products, taking into consideration the diversity of products, their number, and their volume. A very small archive might have only one collection; missions typically have many collections with separate data collections for each instrument, processing level, and mission phase. Follow past practice where possible so that only 'quick-look' products are assigned to the *browse* collection, for example.

The mechanics of defining a collection are straightforward; you create an Inventory table, a specific type of delimited table that lists the logical identifiers (LIDs or LIDVIDs) of all the products that are members of the collection. It also includes a field that identifies each member as either primary (P) or secondary (S). You then create a label that describes the Inventory table.

The Inventory label contains a logical identifier (LID) that uniquely identifies the collection and provides the links between products that share common characteristics. It also contains a version identifier (VID) that distinguishes among several versions of the collection product (if there is more than one).

8.1 Members of a Collection

The members of a collection are designated as being either primary (P) or secondary (S).

- Every product must be a primary member of one and only one collection.
    - That collection is the one in which the product is first registered in the PDS.
- Products already registered in the PDS 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.

8.2 Collection Inventory

The Inventory table is a two-field Table_Delimited digital object where each row (or record) of the table describes one of the member products of the collection.

The first field of each record specifies whether the product is either a primary (P) or secondary (S) member of the collection. The second field specifies either the LID or LIDVID of the product. The only permitted delimiter between the two fields is a comma. The Inventory table file name should be of the form “collection[_*].csv” where the optional base name extension can indicate the type of collection — for example, collection_browse.csv or collection_data_raw.csv.

8.3 Generating and Populating an Inventory Label

Generate a label template using the procedure described in Section 7 of this document. Select Product_Collection for the Root Tag. Two parts of the Inventory label — the Collection area and File_Area_Inventory — will be different from those in basic product labels. In addition, the “description” attribute in the Citation_Information class in the Identification_Area is required. It should provide a terse description (less than 5000 bytes) of the contents of the collection suitable for display in a web browser.

8.3.1 Collection Area

The Collection area contains two attributes – “description”, which is optional, and “collection_type” which is required. The “description” attribute is used to provide an overall description of the collection; although optional, it should be used whenever possible. The value of the collection_type attribute must be one of the following values:

- Browse
- Calibration
- Context
- Data
- Document
- Geometry
- Miscellaneous
- SPICE Kernel
- XML Schema

8.3.2 Citation Information

The Citation_Information class provides information that can be used when citing the collection in journal articles, abstract services, and other reference contexts. This class contains attributes that are used in “searching” for PDS4 collections. A typical Citation_Information looks like this:

```
<Citation_Information>
    <author_list>J. Caesar</author_list>
    <editor_list>A. Smith</editor_list>
    <publication_year>2014</publication_year>
    <keyword>Astrometry</keyword>
    <keyword>Gamma Ray</keyword>
    <keyword>Magnetosphere</keyword>
    <keyword>Voyager</keyword>
```

```
            <keyword>Jupiter</keyword>
            <keyword>ppi-ucla</keyword>
            <description>
                  Collection of browse products in the PPI VG2PLS archive.
            </description>
         </Citation_Information>
```

The "description" and "publication_year" attributes are required; the others are optional but highly recommended since their inclusion improves the PDS4 search process. The "keyword" attribute is especially useful for individually listing terms that may satisfy individual search criteria

8.3.3 Context Area

The Context_Area area contains additional attributes that are used in "searching" for PDS4 products. This is where you provide "reference" information that both describes the collection and links other PDS4 products to the collection. The Context_Area consists of the following optional sub-areas:

1. Time_Coordinates
2. Primary_Result_Summary
3. Investigation_Area
4. Observing_System
5. Target_Information

Try to provide the above information. This is where you document relationships between the collection and external objects having some association with the collection. Future users will then know when the observations were collected, of what, how, and why.

8.3.4 File_Area_Inventory

Similar in structure to File_Area_Observational, this area is more constrained, reflecting the requirements for Inventory table formatting and content. It is required.

File_Area_Inventory specifies the Inventory table through both a File object (which gives the file name and, optionally, the file size, creation date and time, record count, etc.) and an Inventory object, which is a specific form of delimited table, with predefined columns.

In the following example Inventory object description, we have included the required attributes and a few, but not all, of the optional attributes. Almost all of the values are enumerated. The lone exception is <records> the value of which is specific to the Inventory table being described.

```
        <Inventory>
            <offset unit="bytes">0</offset>
            <parsing_standard_id>PDS DSV 1</parsing_standard_id>
            <records>6</records>
            <record_delimiter>Carriage-Return Line-Feed</record_delimiter>
            <field_delimiter>Comma</field_delimiter>
            <Record_Delimited>
                <fields>2</fields>
                <groups>0</groups>
                <maximum_record_length unit="byte">
                   259
                </maximum_record_length>
                <Field_Delimited>
                    <name>Member_Status</name>
                    <field_number>1</field_number>
                    <data_type>ASCII_String</data_type>
                    <maximum_field_length unit="byte">
                       1
                    </maximum_field_length>
                    <description>
                       P indicates primary member of the collection
                       S indicates secondary member of the collection
                    </description>
                </Field_Delimited>
                <Field_Delimited>
                    <name>LIDVID_LID</name>
                    <field_number>2</field_number>
                    <data_type>ASCII_LIDVID_LID</data_type>
                    <maximum_field_length unit="byte">
                       255
                    </maximum_field_length>
                    <description>
                       The LID or LIDVID of a product that is a member
                       of the collection
                    </description>
                </Field_Delimited>
            </Record_Delimited>
            <reference_type>inventory_has_member_product</reference_type>
        </Inventory>
```

The example below shows the entries in a collection inventory table, described by the label fragment above, for a calibrated data collection (data-cal). The first three entries are for primary members, each identified using a LIDVID. The last three entries indicate secondary members – two of which are identified using a LID and one identified by a LIDVID.

```
P,urn:nasa:pds:dph:data-cal:prod1::1.1
P,urn:nasa:pds:dph:data-cal:prod2::1.2
P,urn:nasa:pds:dph:data-cal:prod3::1.1
S,urn:nasa:pds:dph:data-raw:prod1
S,urn:nasa:pds:dph:data-raw:prod2::1.0
S,urn:nasa:pds:dph:data-raw:prod3
```

For further discussion regarding collection products, see the SBN PDS4 Wiki or Section 9C of the PDS4 Standards Reference [2].

9.0 BUNDLES

The product at the highest level in the archive hierarchy is the bundle. Like collections, bundles are 'aggregate' products and consist of a list of references to products. However, in the case of bundles, the referenced products are collection products. PDS does not impose requirements on how bundles are defined except that (1) bundle LIDs must be distinct within the overall holdings of PDS, and (2) each bundle must be approved by a PDS peer review. As part of the design discussion with your consulting DN data engineer (Section 8.0 of this document), you should have determined both how products are organized into collections and how collections are organized intro bundles.

A Product_Bundle identifies all of its member collections, while each Product_Collection identifies all of its basic product members. Unlike collection products, the bundle inventory is embedded within the XML label; it is not a separate file. A Product_Bundle may optionally include a separate 'readme' file.

9.1 Generating and Populating a Bundle Label

Generate and edit a label template for Product_Bundle using the procedure in Section 7 of this document. For the Root Tag select Product_Bundle.

Three areas in a bundle product label differ from those described under basic product labels:

- Bundle
- File_Area_Text
- Bundle_Member_Entry

In addition, the "description" attribute in the Citation_Information class in Identification_Area is required (see Section 8.3.2 for details, which apply to bundles as well as to collections). The description should be a summary of the bundle contents suitable for display in a web browser.

9.2 Bundle Area

The Bundle area contains two attributes – "bundle_type" and "description". This is where you describe the bundle and its contents. If you are creating a bundle that includes a data collection, you will set "bundle_type" to "Archive". If there is no data collection, set "bundle_type" to "Supplemental".

```
<Bundle>
    <description>
        Plasma data from the Voyager 2 encounter with Jupiter.
    </description>
    <bundle_type>Archive</bundle_type>
</Bundle>
```

9.3 File_Area_Text

The File_Area_Text of a bundle product label is used to define the descriptive ‘readme’ file, if one is included. If there is no ‘readme’ file, File_Area_Text should be omitted.

This is an example File_Area_Text label fragment:

```
<File_Area_Text>
     <File>
         <file_name>readme.txt</file_name>
         <local_identifier>README.FILE</local_identifier>
         <creation_date_time>2010-03-12T11:59:04</creation_date_time>
         <file_size unit="byte">22875</file_size>
         <md5_checksum>5ef7af310b99d8189e670830c954a290</md5_checksum>
     </File>
     <Stream_Text>
         <name>VG2 Jupiter plasma bundle</name>
         <local_identifier>BUNDLE.DESCRIPTION</local_identifier>
         <offset unit="byte">0</offset>
         <parsing_standard_id>ASCII text</parsing_standard_id>
         <description>
             Voyager 2 Jupiter plasma bundle description in ASCII text.
         </description>
         <record_delimiter>Carriage-Return Line-Feed</record_delimiter>
     </Stream_Text>
</File_Area_Text>
```

The File class consists of up to seven attributes that describe the ‘readme’ physical file:

- The “file_name” attribute (the only required attribute) provides the name of the ‘readme’ file.
- The “local_identifier” attribute (optional) provides a name for the ‘readme’ file that can be used elsewhere in the label; the value must be unique within the label. It need not have any relationship to “file_name”.
- The “creation_date_time” attribute (optional) provides the date/time when the file was created. The date must be in YMD format; “Z” is an optional suffix.
- The “file_size” attribute (optional) provides the size of the file. The unit (“byte” in this case) must be specified by an XML attribute.
- The “records” attribute (optional, and not used in the example above) provides the number of records in the file.
- The “md5_checksum” attribute (optional) provides the md5 checksum of the file.
- The “comment” attribute (optional, and not used in the example above) provides a brief description of the file.

The Stream_Text class describes how the descriptive information in the ‘readme’ file can be found and extracted; it can include up to seven attributes.

- The “name” attribute (optional) provides a word or combination of words by which the descriptive information is known.

- The “local_identifier” attribute (optional) provides a character string by which the descriptive information can be identified elsewhere in the label; the value must be unique within the XML label.
- The “offset” attribute (required) provides the starting location of the descriptive information within the ‘readme’ file. The unit (“byte” in this case) must be specified by an XML attribute.
- The “parsing_standard_id” attribute (required) provides the name of the standard used for extracting the descriptive information from the byte stream in the file.
- The “object_length” attribute (optional, and not used in the example above) provides the length of the descriptive information. The unit (“byte”, for example) must be specified by an XML attribute.
- The “description” attribute (optional) provides a brief summary of the descriptive information.
- The “record_delimiter” attribute (required) provides the delimiter used to separate lines in the descriptive information. It must be set to “Carriage-Return Line-Feed”.

9.4 Bundle Member Entry

The members of a bundle are specified using the Bundle_Member_Entry class. The Bundle_Member_Entry class is repeated for each collection product in the bundle. For example:

```
<Bundle_Member_Entry>
    <lidvid_reference>
       urn:nasa:pds:example.dph.samplearchive:browse::1.0
    </lidvid_reference>
    <member_status>Primary</member_status>
    <reference_type>bundle_has_browse_collection</reference_type>
</Bundle_Member_Entry>
<Bundle_Member_Entry>
    <lid_reference>
       urn:nasa:pds:example.dph.samplearchive:context
    </lid_reference>
    <member_status>Secondary</member_status>
    <reference_type>bundle_has_context_collection</reference_type>
</Bundle_Member_Entry>
```

In the Bundle_Member_Entry class:

1. You must use either the “lid_reference” or the “lidvid_reference” attribute for each member collection. Collections which are primary members of the bundle may be specified by either a LID or LIDVID depending on which is more appropriate for the particular bundle. Discuss which option to use with your consulting node.

2. The value for “reference_type” depends on the type of collection. It must be one of the following:

• Browse	- bundle_has_browse_collection
• Calibration	- bundle_has_calibration_collection

Context	- bundle_has_context_collection
Data	- bundle_has_data_collection
Document	- bundle_has_document_collection
Geometry	- bundle_has_geometry_collection
Miscellaneous	- bundle_has_member_collection
SPICE Kernel	- bundle_has_spice_kernel_collection
Schema	- bundle_has_schema_collection

3. The value for the “member_status” attribute — “Primary” or “Secondary” — specifies whether the collection is a primary or secondary member of the bundle, respectively.

For further information regarding bundle products, see the SBN PDS4 Wiki.

10.0 DELIVERIES

The organizational products of PDS4 (i.e., bundles and collections) are logical, not physical, structures. However, products transferred to, within, or from PDS, need to be placed into a physical structure. PDS uses delivery "packages" to accomplish such transfers.

In many cases it is convenient to organize delivery packages into a physical structure that parallels the logical structure of the archive (i.e., the bundle product at the root level with subdirectories for collections, etc.). However, alternate organizations are possible (such as flat directory structures for incremental deliveries of accumulating collections). PDS only requires that the sender and receiver agree on the structure in advance and that an MD5 checksum be provided for each file transferred.

10.1 The Package

The term "Package" is the archival material (the data) being transferred. Package software options include ZIP, gzip, and tar; hardware options include thumb drives, external hard drives, and electronic transmission. The data provider and receiver determine the best 'packaging' options for the delivery. In lieu of an ad hoc agreement for transfer, PDS provides a default procedure (see below), which is often recommended by discipline nodes.

10.2 Manifest Files

PDS recognizes two manifest files for transfer packages. The Checksum Manifest is required; the Transfer Manifest is optional. Both are external to the archival material being transferred, meaning they are not considered part of the archive and are not accompanied by label files.

10.2.1 The Checksum Manifest

The "Checksum Manifest" is a specially formatted text file provided with each transfer to, from, or within PDS; it is required for every transfer. The Checksum Manifest contains one record for each file (rather than for each product) in the package. Note that, if the package is a zip file, tar file, etc., the Checksum Manifest contains entries for every file in the package once that package has been unpacked. Currently PDS uses MD5 checksums, so the Checksum Manifest is the output from an MD5 checksum utility such as md5deep. There are two fields in each record:

- the 32 character hexadecimal MD5 checksum value, followed by two spaces
- the file specification name relative to the root directory of the delivery for the file associated with the checksum.

The example below shows the structure of an example Checksum Manifest:

```
4a9a9081a3561e54c12b7e1f6ad4c194  ./CHANGES.TXT
deb4923feb034f6471f44073d3f9496e  ./COPYING.TXT
fc619ce13794aed2f9f362a52b1abc54  ./hashdeep.exe
0326aeff13c17b7f8f5a0917628cab91  ./HASHDEEP.TXT
9432a6dc6bf452bdf3f92e19106d0bbe  ./md5deep.exe
419106c4e9471facb6baf22fa1565643  ./MD5DEEP.TXT
0b5cd51e2de15f532fe75bcfbcd0b924  ./sha1deep.exe
f2c3f93de180d7871fe7b2407434e049  ./sha256deep.exe
81ad20307fdfc959696f31be160db17a  ./tigerdeep.exe
770b6eb6911ca29c83ee2b17e3866fa0  ./whirlpooldeep.exe
```

10.2.2 The Transfer Manifest

A "Transfer Manifest" is a specially formatted Table_Character provided with a transfer to, from, or within PDS; although it is optional in general, it may be required by some discipline nodes. The Transfer Manifest contains one record for each product in the package (including Product_Bundle and each Product_Collection, if included). Each record maps a product LIDVID to the file specification name for that product's XML label file in the transfer package. Note that, if the transfer package is a zip file, tar file, etc., the Transfer Manifest records do the mapping for every label file in the package once the package has been unpacked. The two fields in the table are:

- LIDVID.
- the file specification name (directory path and file name) for the product label associated with the LIDVID. The path is given relative to the root of the transfer package.

The example below shows the structure of a Transfer Manifest:

```
urn:nasa:pds:example.dph.sample:browse:collection::1.0 ./browse/collection.xml
urn:nasa:pds:example.dph.sample:browse:ele_mon::1.0    ./browse/ele_mom_browse.xml
urn:nasa:pds:example.dph.sample:data:collection::1.0   ./data/collection.xml
urn:nasa:pds:example.dph.sample:data:ele_mon::1.0      ./data/ele_mom_data.xml
```

10.3 Transfers Using the PDS4 Default Procedure

If the parties to a transfer do not wish to negotiate an agreement covering the exchange, they may use the default procedure described in the *PDS4 Standards Reference* [2], Section 2B.2. In this case, the data will be transferred using a directory structure that parallels the logical organization of a corresponding bundle — that is, a Product_Bundle at the root, directories with collections below, and basic products in the collection directories. The transfer package itself does not have to meet the requirements for a bundle; in many cases, only part of a bundle (or part of a collection) will be transferred.

With one exception PDS requires that a label and all of its associated digital objects be stored in the same directory. The exception is Product_Document; some of the component files may be stored in subdirectories with respect to the label, in which case attribute directory_path_name provides the path. Because directory_path_name must be valid for both the transfer package and the final storage, close coordination between data provider and the receiving party must be exercised.

For more information and examples, see the *PDS4 Standards Reference* [2], Section 2B.2 and subsections.

10.4 Generating and Populating an XML Label for the "Package"

If a PDS node wishes to register and preserve a delivery package, transfer or checksum manifest, the node must generate an XML label for the resulting product using the appropriate Product_DIP or Product_SIP.

For additional information regarding the definitions and cardinality of the attributes and classes used within the definition of a basic product label, consult the PDS4 Data Dictionary [3].

PART V. LOCAL DICTIONARIES AND OTHER DOCUMENTATION

11.0 LOCAL DATA DICTIONARY

A data dictionary has several purposes. First, the dictionary is a reference for users of the PDS (and other planetary data systems), defining attributes and classes that describe planetary data. Second, the dictionary is a reference for data producers in designing data descriptions. Third, the dictionary ensures that attributes and classes in the data descriptions are used in a standard, consistent, and predictable manner.

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 attributes. For classes, it provides the explicit list of attributes defining the class and indicates which are required, optional, and/or repeatable.

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 discipline schemas.

Data dictionaries are classified as:

- Common
- Discipline specific
- Mission specific

The common dictionary is the fundamental PDS4 dictionary represented in the pds namespace schema and Schematron files. The latter two categories are "local" dictionaries.

The list is intentionally hierarchical. Discipline and mission dictionaries reference the common dictionary and any other relevant discipline dictionaries. However, a mission specific dictionary is not referenced by any other dictionary.

Discipline specific data dictionaries are produced by PDS. These include dictionaries intended to support archives relevant to specific discipline nodes (e.g., an Atmospheres Node dictionary or a Rings Node dictionary) and those that support cross discipline concepts such as geometry and cartography.

Mission specific data dictionaries are produced by PDS and are those that comprise attributes and classes specific to a particular mission or investigation — for example, the Mars 2020 mission could create a data dictionary to define attributes and classes that describe instrumentation and science data that are unique to the Mars 2020 mission. A mission may have a single dictionary or separate the dictionaries by topic — for example having one dictionary for instrumentation descriptors and another for data descriptors. In general, mission personnel manage mission dictionaries; but consulting nodes often participate actively in design and prototyping to ensure conformance with PDS standards.

All PDS archives will be based on the PDS common dictionary; they may also include classes and attributes defined in one or more PDS discipline and/or mission dictionaries.

A local dictionary must reside within a namespace that is unique across all other locally defined dictionaries. It is critical to avoid collisions among namespaces either being used or reserved for use in PDS4 labels. Refer to the *Standards Reference* [2] Section 6B and confer with your PDS consulting node to determine an appropriate namespace if you create a local dictionary. The currently registered namespaces can be found at:

https://pds.nasa.gov/pds4/schema/pds-namespace-registry.pdf

Note that this list is under development; some of the namespaces shown do not conform to the naming constraints in *Standards Reference* Section 6B.2.

For additional information regarding the process for either adding or reserving a new namespace, contact the EN.

11.1 The Mission Area

The Mission_Area, an optional class within the Observation_Area, functionally acts as a container for mission specific classes and attributes, each of which is defined in a local data dictionary. The set of locally defined attributes and classes must be prefixed with a unique mission namespace identifier applicable to the respective dictionary. In the example below `spacecraft_clock_start_count` and `spacecraft_clock_stop_count` have been defined in the dph namespace:

```
<Observation_Area>
       .
       .
       .
  <Mission_Area>
     <dph:spacecraft_clock_start_count>
        1246943630
     </dph:spacecraft_clock_start_count>
     <dph:spacecraft_clock_stop_count>
        1246943631
     </dph:spacecraft_clock_stop_count>
  </Mission_Area>
  <Discipline_Area>
  </Discipline_Area>
</Observation_Area>
```

11.2 The Discipline Area

The Discipline_Area, another optional class within the Observation_Area, functionally acts as a container for discipline specific classes and attributes, each of which is defined in a local data dictionary. The set of locally defined attributes and classes must be prefixed with a unique discipline namespace identifier applicable to the respective dictionary. In the example below `application_process_id` and `application_process_name` are defined in the img namespace:

```
<Observation_Area>
       .
       .
       .
  <Mission_Area>
       .
       .
       .
  </Mission_Area>
  <Discipline_Area>
     <img:application_process_id>
        AGP17
     </img:application_process_id>
     <img:application_process_name>
        Non-linear stretch algorithm 17
     </img:application_process_name>
  </Discipline_Area>
</Observation_Area>
```

11.3 Building and Using Local Data Dictionaries

PDS can merge the attributes and classes defined in a local data dictionary with the PDS4 Information Model. Merging ensures that your locally defined attributes and/or classes will be contained in future builds of the PDS4 Data Dictionary and generic mission and discipline schemas. Merging is optional but, once completed, means that your locally defined classes and attributes are readily available any time you need them. A successful merge requires, however, that the local definitions be compatible with the common structure and contents. PDS provides LDDTool to facilitate this process.

The input to LDDTool is an XML file based on the Ingest_LDD class in the common schema. As producer of the local dictionary, you generate and populate the input file, run LDDTool, and submit both the input file and the output files to PDS. Details of this process are contained in the *Ingest LDD Users Guide*, which is included in the zip file containing LDDTool. We summarize key steps below.

Note that the following discussion uses oXygen as the vehicle for demonstrating the local-DD process. In no way does this suggest or imply a recommendation or endorsement for using oXygen over any of the other XML-aware editors (XAEs).

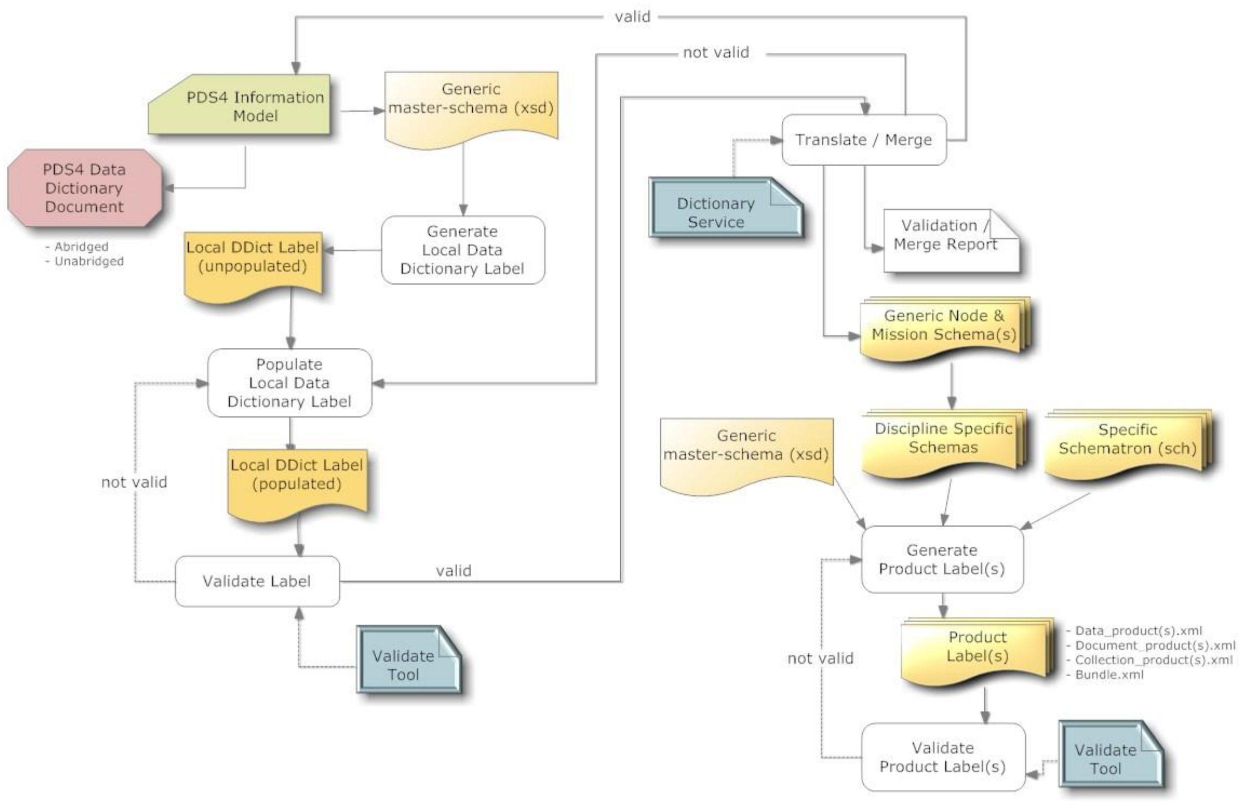

valid
not valid
PDS4 Information Model
Generic master-schema (xsd)
PDS4 Data Dictionary Document
- Abridged
- Unabridged
Local DDict Label (unpopulated)
Generate Local Data Dictionary Label
Populate Local Data Dictionary Label
Local DDict Label (populated)
not valid
Validate Label
valid
Validate Tool
Translate / Merge
Dictionary Service
Validation / Merge Report
Generic Node & Mission Schema(s)
Generic master-schema (xsd)
Discipline Specific Schemas
Specific Schematron (sch)
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 11-1. Local data dictionary development and label production flow diagram.

Step #1: Download the current version of the common-schema to a directory/folder on your local computer. Files will have names of the form PDS4_PDS_nnnn.xsd and can be found at:

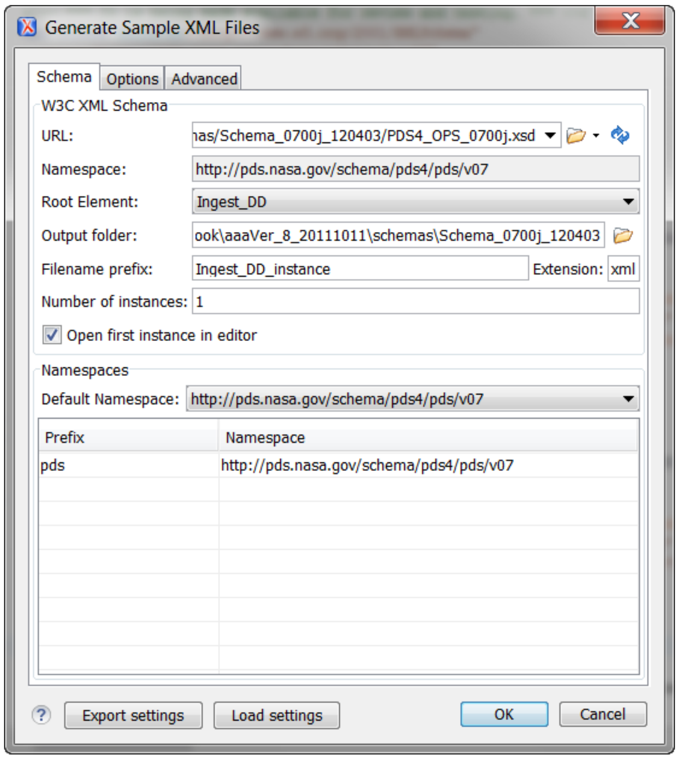
http://pds.nasa.gov/pds4/schema/released/pds/

Step #2: Using the XML-aware editor (XAE) of your choice, open the common-schema (XSD) file. Locate the Ingest_LDD class. This is the class from which we will generate the input XML file which the dictionary tool will use to generate the schema for your dictionary. The process for producing this input file is the same as the process for producing a label template file (see Appendix D of this document). In this case instead of using Product_* in your Root Tag, you use Ingest_LDD.

Step #3: Most XML-aware editors provide a capability to generate an XML-template from a schema.

In oXygen:

- select: Tools/Generate Sample XML Files.



W3C XML Schema

URL: ıas/Schema_0700j_120403/PDS4_OPS_0700j.xsd

Namespace: http://pds.nasa.gov/schema/pds4/pds/v07

Root Element: Ingest_DD

Output folder: ook\aaaVer_8_20111011\schemas\Schema_0700j_120403

Filename prefix: Ingest_DD_instance Extension: xml

Number of instances: 1

☑ Open first instance in editor

Namespaces

Default Namespace: http://pds.nasa.gov/schema/pds4/pds/v07

Prefix	Namespace
pds	http://pds.nasa.gov/schema/pds4/pds/v07

Figure 11-2. First form in generating template file

You should see a form like that shown in Figure 11-2; notice that the "Schema" tab is active. This form is used to specify the XSD from which the template file will be generated and to specify the output template file name and location. Fill in the boxes as follows:

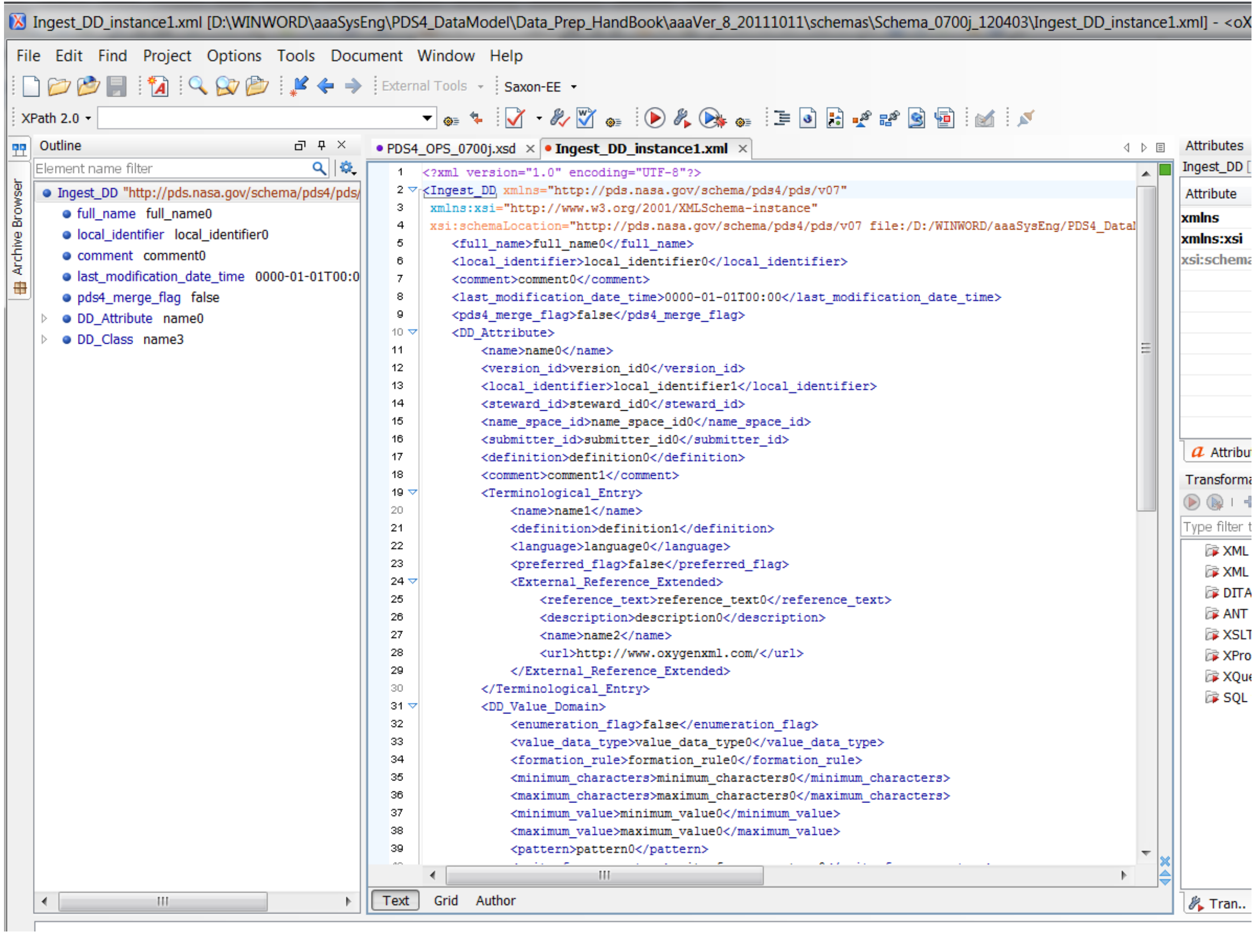
- In “URL” use the ‘folder’ icon to select the local copy of the common-schema file.
- In “Namespace” the value should have been auto-populated with the value specified in the common-schema. The value should be fairly close to the value displayed above.
- In “Root Element” use the drop-down menu to select the product for which you want to create a template file – Ingest_LDD.
- In “Output folder” specify the directory/folder where you want the template file to reside (when created).
- In “Filename prefix” specify the name of the template file.
- In “Number of instances” enter “1”. If you enter a larger number, additional copies of the template will be created, each having a name beginning with the value specified in "Filename prefix".
- Check the “Open first instance in editor” box so that the template file will be displayed in oXygen when created.
- “Default Namespace” should have been auto-populated with the value specified in the common-schema. The value should be fairly close to the value displayed above.

- The “Prefix” and “Namespace” areas should have been auto-populated with values fairly close to those displayed above.

Once you have verified the above values, select the "Options" tab (at the top of the dialog box) and fill in the boxes on the next form:

- Check the “Generate optional elements” box so all optional elements are included in the template file.
- Check the “Generate optional attributes” box so all optional attributes are included in the tenplate file.
- In “Values of elements and attributes” select "Default" using the pull-down menu.
- In “Preferred number of repetitions” specify "1" so that a single instance of each class and attribute will be included in the template file.
- In “Maximum recursivity level” specify "1" as the maximum depth in case of recursivity.
- In “Choice strategy” select "Random" using the pull-down menu.

Once, you have verified the above values, click the "OK" button. This will generate the XML template file, which should be displayed in your XAE window (Figure 11-3).


Ingest_DD_instance1.xml [D:\WINWORD\aaaSysEng\PDS4_DataModel\Data_Prep_HandBook\aaaVer_8_20111011\schemas\Schema_0700j_120403\Ingest_DD_instance1.xml] - <oX
File Edit Find Project Options Tools Document Window Help
External Tools
Saxon-EE
XPath 2.0
Outline
Element name filter
Archive Browser
Ingest_DD "http://pds.nasa.gov/schema/pds4/pds/
full_name full_name0
local_identifier local_identifier0
comment comment0
last_modification_date_time 0000-01-01T00:0
pds4_merge_flag false
DD_Attribute name0
DD_Class name3
PDS4_OPS_0700j.xsd
Ingest_DD_instance1.xml
Text
Grid
Author
Attributes
Ingest_DD
Attribute
xmlns
xmlns:xsi
xsi:schema
Attribu
Transform
Type filter
XML
XML
DITA
ANT
XSLT
XPro
XQu
SQL
Tran..

```
<?xml version="1.0" encoding="UTF-8"?>
<Ingest_DD xmlns="http://pds.nasa.gov/schema/pds4/pds/v07"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://pds.nasa.gov/schema/pds4/pds/v07 file:/D:/WINWORD/aaaSysEng/PDS4_Data
    <full_name>full_name0</full_name>
    <local_identifier>local_identifier0</local_identifier>
    <comment>comment0</comment>
    <last_modification_date_time>0000-01-01T00:00</last_modification_date_time>
    <pds4_merge_flag>false</pds4_merge_flag>
    <DD_Attribute>
        <name>name0</name>
        <version_id>version_id0</version_id>
        <local_identifier>local_identifier1</local_identifier>
        <steward_id>steward_id0</steward_id>
        <name_space_id>name_space_id0</name_space_id>
        <submitter_id>submitter_id0</submitter_id>
        <definition>definition0</definition>
        <comment>comment1</comment>
        <Terminological_Entry>
            <name>name1</name>
            <definition>definition1</definition>
            <language>language0</language>
            <preferred_flag>false</preferred_flag>
            <External_Reference_Extended>
                <reference_text>reference_text0</reference_text>
                <description>description0</description>
                <name>name2</name>
                <url>http://www.oxygenxml.com/</url>
            </External_Reference_Extended>
        </Terminological_Entry>
        <DD_Value_Domain>
            <enumeration_flag>false</enumeration_flag>
            <value_data_type>value_data_type0</value_data_type>
            <formation_rule>formation_rule0</formation_rule>
            <minimum_characters>minimum_characters0</minimum_characters>
            <maximum_characters>maximum_characters0</maximum_characters>
            <minimum_value>minimum_value0</minimum_value>
            <maximum_value>maximum_value0</maximum_value>
            <pattern>pattern0</pattern>
```

Figure 11-3. Template input file for LDDTool as generated by an oXygen XML aware editor.

If you are using the oXygen XAE, ensure that you can see a little green box in the upper right. If the box is red (or not green), the template is not valid; contact your consulting DN for suggestions on resolving discrepancies.

The template, once populated with real values, will be used by LDDTool to generate your local dictionary product (schema, Schematron, and XML label files).

Step #4: With the template displayed in your XAE, you are now ready to begin populating it with the metadata associated with each attribute and class that is to be defined in the local data dictionary.

As you populate the template, ensure that the XML remains 'fully formed' — that is, keep the box in the upper right green. If the color changes, you have introduced an error in the XML. oXygen will do this validation automatically as you work; other XAEs may have similar real-time validations (or not). You can also use the PDS4 Validation Tool to validate the template.

```
<?xml version="1.0" encoding="UTF-8"?>
<Ingest_LDD
  xmlns="http://pds.nasa.gov/pds4/pds/v1"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://pds.nasa.gov/pds4/pds/v1 file:/TEST/PDS4_PDS_1401.xsd">
    <name>name0</name>
    <ldd_version_id>ldd_version_id0</ldd_version_id>
    <full_name>full_name0</full_name>
    <steward_id>steward_id0</steward_id>
    <namespace_id>namespace_id0</namespace_id>
    <external_property_maps_id>
       external_property_maps_id0
    </external_property_maps_id>
    <comment>comment0</comment>
    <last_modification_date_time>0000</last_modification_date_time>

    <DD_Attribute>
    ...
    </DD_Attribute>

    <DD_Class>
    ...
    </DD_Class>

    <DD_Rule>
    ...
    </DD_Rule>

    <Property_Maps>
    ...
    </Property_Maps>
  </Ingest_LDD>
```

Your unedited template should look something like the above. It has a standard XML Declaration, a Root Tag, some dictionary overview attributes, and four classes defining the types of content that can be included in the dictionary: DD_Attribute, DD_Class, DD_Rule, and Property_Maps.

Step #5: As necessary, edit the XML Declaration and Root Tag. An XML tag is a character string delimited by a pair of angle brackets — "<" and ">" (Section 7.2 of this document).

1. The XML Declaration (line 1 in the example above) needs no editing.
2. The Root Tag correctly references the PDS common dictionary and the XML Schema Instance (xsi) namespace, so lines 2-4 need no editing.
3. If your local dictionary will reference any other discipline dictionary, you need to add that namespace identfication to the Root Tag. For purposes of this example, we will assume you will reference the geometry discipline dictionary. The namespace ID for the geometry dictionary is "geom".
4. You need to add a reference to the Schematron file for each of the dictionaries you plan to reference, including the common dictionary. Do this by inserting one XML tag for each dictionary between the XML Declaration and the Root Tag. As discussed in Section 7.2.2, XAEs have different requirements regarding Schematron references; check your XAE user guide or contact your consulting DN if there are problems.

After these changes, your XML Prolog and Root Tag should look like this, where the new lines are shown in red:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-model href="http://pds.nasa.gov/pds4/pds/v1/PDS4_PDS_1401.sch"?>
<Ingest_LDD
  xmlns="http://pds.nasa.gov/pds4/pds/v1"
  xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
  xmlns:geom=http://pds.nasa.gov/pds4/geom/v1
  xsi:schemaLocation="http://pds.nasa.gov/pds4/pds/v1 file:/TEST/PDS4_PDS_1401.xsd">
```

5. The next edit is for the line beginning "xsi:schemaLocation". In the unedited template, the entry refers to a local file. As discussed in Section 7.2, the final version should not include local references, although some XAEs make this difficult. Here we assume the use of a catalog file to enable the XAE to reference a local version, while the edited template provides the permanent URI for the schema. The unedited version refers to the common schema; in this example we also need to refer to the geometry schema. The final XML Prolog and Root Tag is:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-model href="http://pds.nasa.gov/pds4/pds/v1/PDS4_PDS_1401.sch"?>
<?xml-model href="http://pds.nasa.gov/pds4/geom/v1/PDS4_GEOM_1401.sch"?>
<Ingest_LDD
  xmlns="http://pds.nasa.gov/pds4/pds/v1"
  xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
  xmlns:geom=http://pds.nasa.gov/pds4/geom/v1
  xsi:schemaLocation="http://pds.nasa.gov/pds4/pds/v1
                      http://pds.nasa.gov/pds4/pds/v1/PDS4_PDS_1401.xsd
                      http://pds.nasa.gov/pds4/geom/v1
                      http://pds.nasa.gov/pds4/geom/v1/PDS4_GEOM_1401.xsd">
```

Step #6: Substitute real values for the placeholders in lines beginning <name> through <last_modification_date_time>.

<name> is the name of your local dictionary (we will use “Phoenix Mission Dictionary” here).
<ldd_version_id> is the version ID of your dictionary. See the *Standards Reference* [2], Section 6D.3.2 for a complete discussion of dictionary and namespace version IDs. During development, the first position may be “0”. The first official delivery to the PDS should have a “1” in the first position.
<full_name> is the full name of the person responsible for developing and maintaining the dictionary.
<steward_id> is the identifier for the agency which is the steward for the dictionary. Discuss this with your consulting node.
<namespace_id> the namespace ID for your dictionary. You may propose an ID to your consulting node; but, in order to ensure no conflicts with existing IDs, the final determination is made by the consulting node and the Engineering Node. Contact your consulting node early to avoid having to redo work later.
<external_property_maps_id> Contact your consulting node.
<comment> We strongly recommend you use this attribute to describe your initial release and to document changes in subsequent versions.
<last_modification_date_time> The approximate date and time of the most recent changes.

Step #7: The unedited template contains one copy of this DD_Attribute class definition (below).

```
    <DD_Attribute>
        <name>name1</name>
        <version_id>0.0</version_id>
        <local_identifier>local_identifier0</local_identifier>
        <nillable_flag>false</nillable_flag>
        <submitter_name>submitter_name0</submitter_name>
        <definition>definition0</definition>
        <comment>comment1</comment>
        <Internal_Reference>
            <lidvid_reference>lidvid_reference111</lidvid_reference>
            <reference_type>reference_type0</reference_type>
            <comment>comment2</comment>
        </Internal_Reference>
        <Terminological_Entry>
            <name>name2</name>
            <definition>definition1</definition>
            <language>language0</language>
            <preferred_flag>false</preferred_flag>
            <instance_id>instance_id0</instance_id>
            <External_Reference_Extended>
                <doi>doi0</doi>
                <reference_text>reference_text0</reference_text>
                <description>description0</description>
                <name>name3</name>
                <url>http://www.oxygenxml.com/</url>
            </External_Reference_Extended>
        </Terminological_Entry>
        <DD_Value_Domain>
            <enumeration_flag>false</enumeration_flag>
            <value_data_type>value_data_type0</value_data_type>
            <formation_rule>formation_rule0</formation_rule>
            <minimum_characters>minimum_characters0</minimum_characters>
            <maximum_characters>maximum_characters0</maximum_characters>
            <minimum_value>minimum_value0</minimum_value>
            <maximum_value>maximum_value0</maximum_value>
            <pattern>pattern0</pattern>
            <unit_of_measure_type>
               unit_of_measure_type0
            </unit_of_measure_type>
            <specified_unit_id>specified_unit_id0</specified_unit_id>
            <DD_Permissible_Value>
                <value>value0</value>
                <value_meaning>value_meaning0</value_meaning>
                <Terminological_Entry>
                    <name>name4</name>
                    <definition>definition2</definition>
                    <language>language1</language>
                    <preferred_flag>false</preferred_flag>
                    <instance_id>instance_id1</instance_id>
                    <External_Reference_Extended>
                        <reference_text>reference_text1</reference_text>
                    </External_Reference_Extended>
                </Terminological_Entry>
            </DD_Permissible_Value>
        </DD_Value_Domain>
    </DD_Attribute>
```

You will need a complete copy for each attribute you include in your dictionary. However, before you do several dozen "copy and paste" operations, look closely at the contents. Several of the attributes in this class will have the same value in all definitions. Edit these first, and then do the copy and paste. Candidate attributes for copying include version_id (since they all probably start with "1.0") and submitter_name, which is probably the same for most (if not all) of the attributes – usually the individual doing the typing.

All of the attributes must be defined before any of the classes in the dictionary can be defined. For specific information on how to populate each DD_Attribute class, see the *LDD Tool Users Guide* and the SBN PDS4 Wiki. Do not hesitate to contact your consulting node.

Step #8: The template sections for DD_Class, DD_Rule, and Property_Map classes can be replicated and edited once you have the DD_Attribute definitions completed. Of course, the process of developing classes and attributes is iterative; you may discover that you need another attribute while you are fleshing out the classes. But you should not end up with attributes that haven't been used and you should never have a class with an attribute that has not been defined. By referencing the PDS common dictionary and selected discipline dictionaries, you can import both attributes and classes (and their definitions) into your own dictionary.

Replicate and edit the template sections for DD_Class, DD_Rule, and Property_Map classes as you did for DD_Attribute; identify values that will not change across your dictionary, edit those parts of the template, and then do your “copy and paste” operation before buckling down to the detail of populating the individual definitions. You can find details in the *LDD Tool Users Guide*, and the SBN PDS4 Wiki. Do not hesitate to contact your consulting node.

Step #9: At this point, you should have a fully formed (populated) and compliant XML data dictionary input file. Run LDDTool. See the *LDD Tool Users Guide* for instructions on how to run the tool and what command line flags to set.

Step #10: Review the output files — especially log files — to identify and correct errors. Obtain assistance from your consulting node as necessary. Then repeat Step #9 until you have an error-free run.

Step #11: Once you have an error-free LDDTool run, you should send the input XML file and the output files to the PDS Engineering Node. PDS/EN will use software to “ingest” the classes and attributes that you defined (in Ingest_LDD.xml) into the IM. EN will then send back to you the following:

1. A local-dictionary XSD that contains the classes and attributes that you defined.
2. A validation report that indicates either “success” or issues that may still need to be resolved.
3. A local-dictionary HTML file that can be browsed for a human-readable dictionary content.

The successful result is that you will have online access to your local dictionary schema (XSD) at:

`https://pds.nasa.gov/pds4/schema/released/`

12.0 OTHER DOCUMENTATION

Supplementary or ancillary reference materials are included with archive products to improve their short- and long-term usability. These documents augment the information furnished in the product labels and provide further assistance in understanding the data products and accompanying materials. Typical archive documents include:

- Flight project documents
- Instrument papers
- Science articles
- Software Interface Specifications (SISs)
- Software user manuals

Criteria for inclusion of a document in a PDS archive are:

1. The document is necessary for evaluation, understanding, and use of the data.
2. If not ‘necessary’, the document is at least useful.
3. Document distribution is not restricted.

You should also consider how documentation should be divided among internal and external documents.

PDS takes the requirement for documentation very seriously. It must be possible for a scientist familiar with the field (but not necessarily with the observing system or data) to understand and use the data based on information contained in labels, documents within the archive, and external documents referenced from the archive. In general, the PDS seeks to err on the side of completeness.

The Product Document and Product_File_Text classes may be used to describe document products, which are delivered as part of a document collection. Each document must be saved in a PDS-compliant format — flat UTF-8 text, PDF/A-1a (which is preferred), or PDF/A-1b. Documents prepared for inclusion in an archive must be conformant to the ‘Policy on Formats for PDS4 Data and Documentation’ at

http://pds.nasa.gov/policy/format_policies_final.pdf

Documents prepared for inclusion in an archive are expected to meet not only the PDS label and format requirements, but also the structural, grammatical and lexical requirements of a refereed journal submission. Documents submitted for archiving which contain spelling errors, poor grammar or illogical organization may be rejected and may ultimately lead to the rejection of the submitted data for lack of adequate documentation.

12.1 Internal versus External Documentation

To ensure integrity of the archive, the preferred approach is to have all documentation within the archive. In some cases, however, this is either impractical or impossible. For example, when a copyright holder refuses re-use, the copyrighted material must be referenced through an outside source.

The Reference_List class may be used to reference relevant material that resides either inside or outside PDS; it may be associated with any of over two dozen different product classes including Product_Observational, Product_Context, Product_Document, and Product_Collection.

The Internal_Reference class may be used to establish relationships with material inside PDS using a LID or LIDVID either through Reference_List or directly, such as from Observing_System_Component. The example below shows how an Instrument_Host context product cites documentation in an Investigation context product.

```
<Product_Context>
      .
      .
      .
      <Instrument_Host> ... </Instrument_Host>
      <Reference_List>
            <Internal_Reference>
                  <lidvid_reference>
                      urn:nasa:pds:context:investigation:mission.voyager::1.0
                  </lidvid_reference>
                  <reference_type>
                      instrument_host_to_investigation
                  </reference_type>
            </Internal_Reference>
      </Reference_List>
            .
            .
            .
</Product_Context>
```

The External_Reference class provides a complete bibliographic citation to a published work outside PDS, optionally including its Digital Object Identifier (DOI) and a description of the work. External_Reference may be used through Reference_List or directly, such as from Observing_System_Component. The example below shows how External_Reference could be used in place of Internal_Reference in the previous example.

```
<Product_Context>
      .
      .
      .
      <Instrument_Host> ... </Instrument_Host>
      <Reference_List>
            <External_Reference>
                  <reference_text>
                        Morrison, D., Voyages to Saturn, NASA SP-451,
                        227 p., National Aeronautics and Space
                        Administration, Washington, DC, 1982.
                  </reference_text>
            </External_Reference>
      </Reference_List>
            .
            .
            .
</Product_Context>
```

12.2 Restricted Documents

Certain documents have restricted distributions so cannot be included in PDS archives, which are accessible to anyone with an internet connection. The most common situations involve documents protected by copyright (when the copyright holder has not given permission for re-use) and documents covered by United States International Traffic in Arms Regulations (ITAR). Both circumstances are covered by United State law, and violations could lead to serious consequences for both PDS and the individuals involved. Discuss the issues with your consulting DN should it appear that your documents may be restricted; your consulting DN may wish to raise the questions to higher levels within PDS and NASA.

Copyright issues are the easiest to resolve. If the document(s) in question are available to the public through the copyright holder, then External_Reference is appropriate and the data user can negotiate access to the documents with the copyright holder, possibly in exchange for a fee.

ITAR covers a wide range of hardware, software, and operations topics in the weapons, missles, and technology areas. In general, the results of science investigations are not subject to ITAR control; but the performance, design, and operation of the equipment may be. Construction and operational details of new, state-of-the-art instrumentation are likely candidates for review. NASA centers (e.g., JPL and GSFC) and institutions heavily involved in technology (JHU/APL) often have ITAR review teams which can determine whether questionable documents are suitable for release and certify them as such. If not suitable, investigation teams may have to rewrite the documents; determining when redaction has a negative impact on the ability of a science user to understand and work with the data can be challenging.

In the end it is the responsibility of the data provider to submit unrestricted documentation (which may include references to material that is external to PDS but publicly accessible) that is sufficient for a science user to understand and work with the data.

PART VI. VALIDATE THE ARCHIVE

13.0 VALIDATION

This section describes the process of validating your PDS4 label design and the relationships among the common-schema, the common-Schematron file, the discipline specific and mission specific schema and Schematron files, and the resulting XML label file(s).

Figure 13-1 illustrates the output stage of a data production pipeline in which a data provider ensures that XML documents are compliant with the parent schemas. The validation process checks that each label is valid when compared against

a. the common-schema
b. discipline specific and mission specific schemas, if provided
c. Schematron files, if provided.

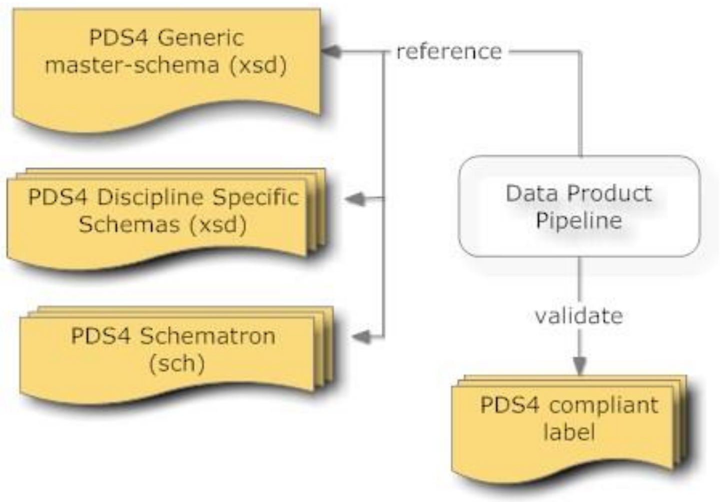


Figure 13-1. Diagram showing validation of product labels as a process at the end of a data production pipeline. Mission specific schemas are not shown explicitly; they would appear at the same location as discipline specific schemas.

13.1 Data Product Validation

Prior to delivery to PDS, data providers must ensure that their data products, collections, and bundles are compliant with PDS standards [2]. Compliance with PDS standards includes syntactic, semantic, content and referential integrity. It is suggested that PDS supplied tools be run prior to delivery.

13.2 PDS4 Validation Tool

The PDS4 Validate Tool is used for validating PDS4 product labels and product data. The associated specific schema for the product label specifies syntactic and semantic constraints.

PDS provides a suit of PDS4 tools which can be found here:

http://pds.nasa.gov/pds4/software/index.shtml

The PDS4 Validate Tool can be found here:

http://pds.nasa.gov/pds4/software/validate

APPENDIX A ACRONYMS AND ABBREVIATIONS

The following acronyms and abbreviations pertain to this document:

APL	Applied Physics Laboratory
CSV	Comma-Separated Values
DIP	Dissemination Information Package
DN	Discipline Node (PDS)
DPH	Data Providers' Handbook
DTD	Document Type Definition
EN	Engineering Node (PDS)
GIF	Graphics Interchange Format
GSFC	Goddard Space Flight Center (NASA)
ITAR	International Traffic in Arms Regulations
JHU	Johns Hopkins University
JPEG	Joint Photographic Experts Group
JPL	Jet Propulsion Laboratory
LDD	Local Data Dictionary
LID	Logical Identifier
LIDVID	Versioned LID
NASA	National Aeronautics and Space Administration
P	Primary (member)
PDF	Portable Document Format
PDF/A	Portable Document Format (archival)
PDS	Planetary Data System
PDS3	PDS version 3
PDS4	PDS version 4
PNG	Portable Network Graphics
PPI	Planetary Plasma Interactions (PDS DN)
S	Secondary (member)\
SBN	Small Bodies Node (PDS)
SCH	Schematron
TIFF	Tagged Image File Format
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
URN	Uniform Resource Name
XAE	XML aware editor
XML	eXtensible Markup Language
Xpath	XML Path Language
XSD	XML Schema Definition

APPENDIX B SCHEMATRON REFERENCES

In markup languages, **Schematron** is a rule-based validation language for making assertions about the presence or absence of patterns in XML trees. It is a structural schema language expressed in XML using a small number of XML elements and XPath.

Schematron is capable of expressing constraints in ways that other XML schema languages like XML Schema and DTD cannot. For example, it can require that the content of an XML element be controlled by one of its siblings. Or it can request or require that the root element, regardless of what element that is, must have specific attributes. Schematron can also specify required relationships between multiple XML files.

Constraints and content rules may be associated with "plain-English" validation error messages, allowing translation of numeric Schematron error codes into meaningful user error messages.

The following example Schematron “rule” ensures the value for the information_model_version attribute in the Identification_Area exactly matches ‘1.0.0.0’ or an error message is generated:

```
<sch:pattern>
      <sch:rule context=”pds:Identification_Area”>
      <!-- ================================================ -->
      <!-- Test: ensure 'information_model_version' value     -->
      <!--       matches expected-value                       -->
      <!-- ================================================ -->
      <sch:assert test="pds:information_model_version='1.0.0.0'">
            Identification_Area.information_model_version: does
            NOT specify the current version.
      </sch:assert>
      </sch:rule>
</sch:pattern>
```

The following are good sources of additional information:

- Roger Costello has posted a good set of introductory tutorials. There is no need to run through all of the tutorials. But “Two Types of XML Validation”, “Usage and Features”, and “Overview” will set the stage for what Schematron is.
    - http://www.xfront.com/schematron/index.html

- Miloslav Nic has posted a great set of starter examples and ‘how to’ information at:
    - http://zvon.org/xxl/SchematronTutorial/General/toc.html

- You can download and walk through “A hands-on introduction to Schematron” by Uche Ogbuji from:

    - https://www6.software.ibm.com/developerworks/education/x-schematron/x-schematron-a4.pdf

APPENDIX C XML CATALOG FILES

XML documents typically refer to external entities — for example, schemas for the Document Type Definition. These external relationships are expressed using URIs, typically given as URLs.

However, if your locations are given as remote URLs, they only work when your network can reach them. Relying on remote resources makes XML processing susceptible to your connectivity choices and to both planned and unplanned network outages.

Conversely, if you use local URLs, they are only useful in the context where they were initially created. For example, the URL "../../xml/dtd/docbookx.xml" will not work once your labels have been ingested into the PDS system and downloaded to someone else's computer.

One way to avoid these problems is to use an entity resolver or a URI Resolver. A resolver can examine the URIs of the resources being requested and determine how best to satisfy those requests. The XML 'catalog' file is one such resolver; it is a document describing a mapping between external entity references and locally-cached equivalents.

The following is an example of a XML catalog file that resolves the locations of schemas as the schemas follow a typical progression from development (DEV) through local archive (ARCHIVE), to fully ingested (ONLINE) status.

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog">
  <!-- ======================================================= -->
  <!-- 1st reference is to DEV instance                        -->
  <!-- ======================================================= -->
    <group xml:base="file:/D:/DEV/schemas/">
    <uri name=“http://pds.nasa.gov/pds4/pds/v1″ uri="PDS4_PDS_1200.xsd"/>
    <uri name="http://pds.nasa.gov/schema/pds4/phxmd/v01" uri="PHXMD_1200.xsd"/>
    <system systemId="PDS4_PDS_1200.xsd" uri="PDS4_PDS_1200.xsd"/>
  </group>

  <!-- ======================================================= -->
  <!-- 2nd reference is to ARCHIVE instance                    -->
  <!-- ======================================================= -->
  <group xml:base="file:/D:/ARCHIVE/schemas/">
    <uri name="http://pds.nasa.gov/pds4/pds/v1" uri="PDS4_PDS_1200.xsd"/>
    <uri name="http://pds.nasa.gov/schema/pds4/phxmd/v01" uri="PHXMD_1200.xsd"/>
    <system systemId="PDS4_PDS_1200.xsd" uri="PDS4_PDS_1200.xsd"/>
  </group>

  <!-- ======================================================= -->
  <!-- 3rd reference is to ONLINE instance                     -->
  <!-- ======================================================= -->
  <system systemId="http://pds.nasa.gov/schema/pds4/phxmd/v01"
uri="http://pds.jpl.nasa.gov/schemas/PHXMD_1200.xsd"/>
  <uri name="http://pds.nasa.gov/schema/pds4/phxmd/v01"
uri="http://pds.jpl.nasa.gov/schemas/PHXMD_1200.xsd"/>

</catalog>
```

Once the XML catalog file has been created, it will need to be ‘registered’ with the XML-aware editor that you are using (note that not all XML-aware editors accommodate catalog files, and the ones that do may interact with them differently). Once registered, you will also need to ensure that the XML catalog file is being referenced by your XML product labels.

APPENDIX D: CREATING A LABEL TEMPLATE

This appendix lists the steps to be used in creating a label template. There are several ways to do this; they are described in separate subsections below.

D.1 Appropriation

The simplest way to obtain a label template is to copy the label from a product that is similar to the one you are planning to label. Your consulting DN data engineer may be able to provide one.

D.2 Eclipse XML Editor

The Small Bodies Node wiki includes instructions on generating new label files from the common-schema using the Eclipse XML editor. Go to:

http://borrelly.astro.umd.edu/wiki/Eclipse:_Creating_a_New_XML_File_from_an_XSD_Schema_File

The SBN wiki pages cover editing of the XML Declaration and Root Tag, so you can skip the corresponding parts of Section 7 in this document.

D.3 oXygen XML Editor

This set of steps assumes that you are using the oXygen 17.0 XML editor (other versions of oXygen should perform similarly). It produces a template from the common-schema (xsd file); the template has all levels and all options. You must then remove options you don't want by subsequent editing. XML elements for attributes are created with empty values; but you can select an option which inserts values which are either valid or not (your choice). You then need to edit those to legitimate values (or placeholders) for your labels.

Step 1: Download a copy of the current common-schema (xsd)

a) The current version of the PDS4 common-schema (xsd) can always be found at the PDS4 web page. Go to:

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

b) Assuming you want the current pair of released files (for Information Model n.n.n.n), click on

- The current Schematron file: PDS4_PDS_nnnn.sch
  and
- The current common-schema file: PDS4_PDS_nnnn.xsd

c) Save both files to a working directory or folder.

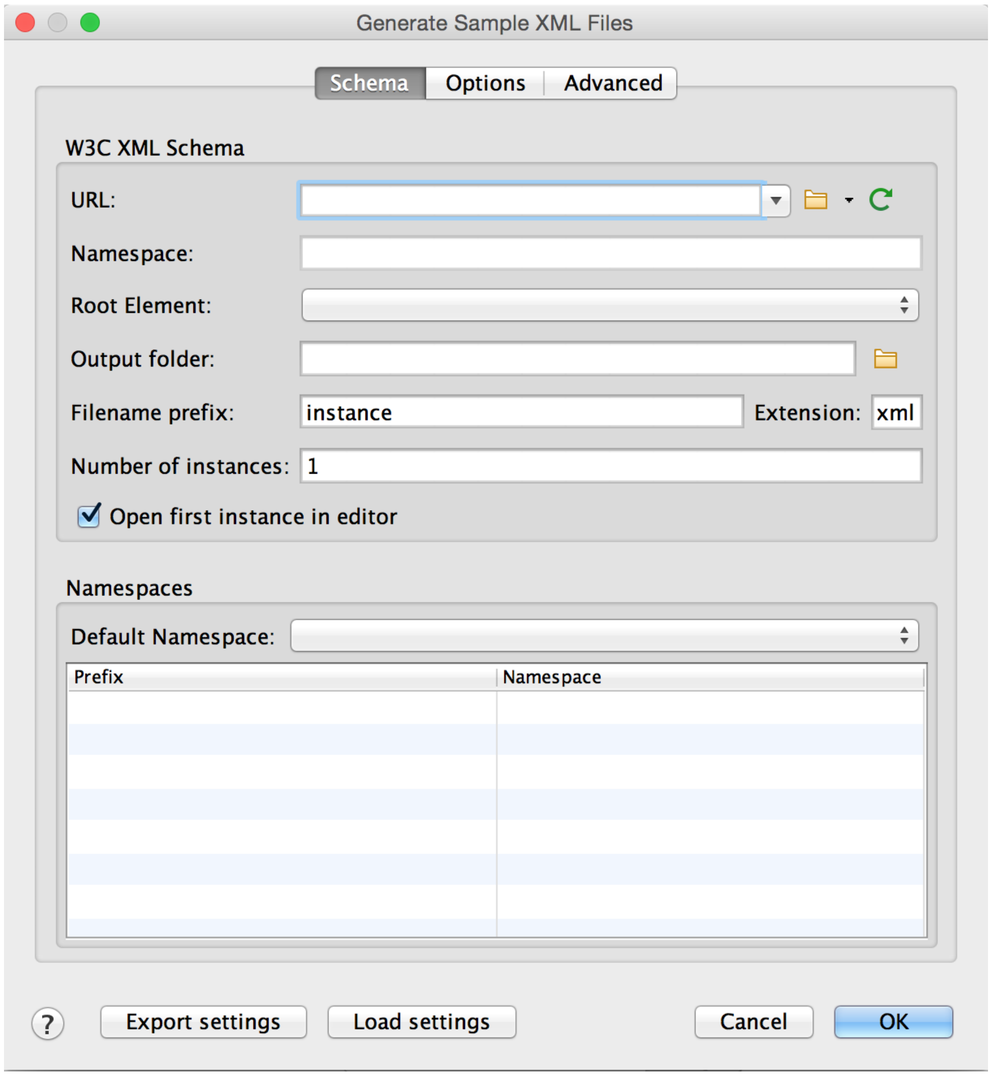
Step 2: Open the common-schema in an XML-aware Editor

a) Using the oXygen XML editor, open the common-schema (xsd) file in your working directory.

b) Pull down the “Tools” menu from the top menu bar and select “Generate Sample XML Files …” You will get a window like the one shown in Figure D-3.

c) From the three tabs at the top, select the “Schema” tab.

d) The “URL” entry will be blank. Use the folder icon at the right to select the local copy of the common-schema (the xsd file you just downloaded). This will also populate the “Namespace” with the appropriate value from the xsd file.



URL:
Namespace:
Root Element:
Output folder:
Filename prefix: instance Extension: xml
Number of instances: 1
☑ Open first instance in editor
Default Namespace:

Figure D-3

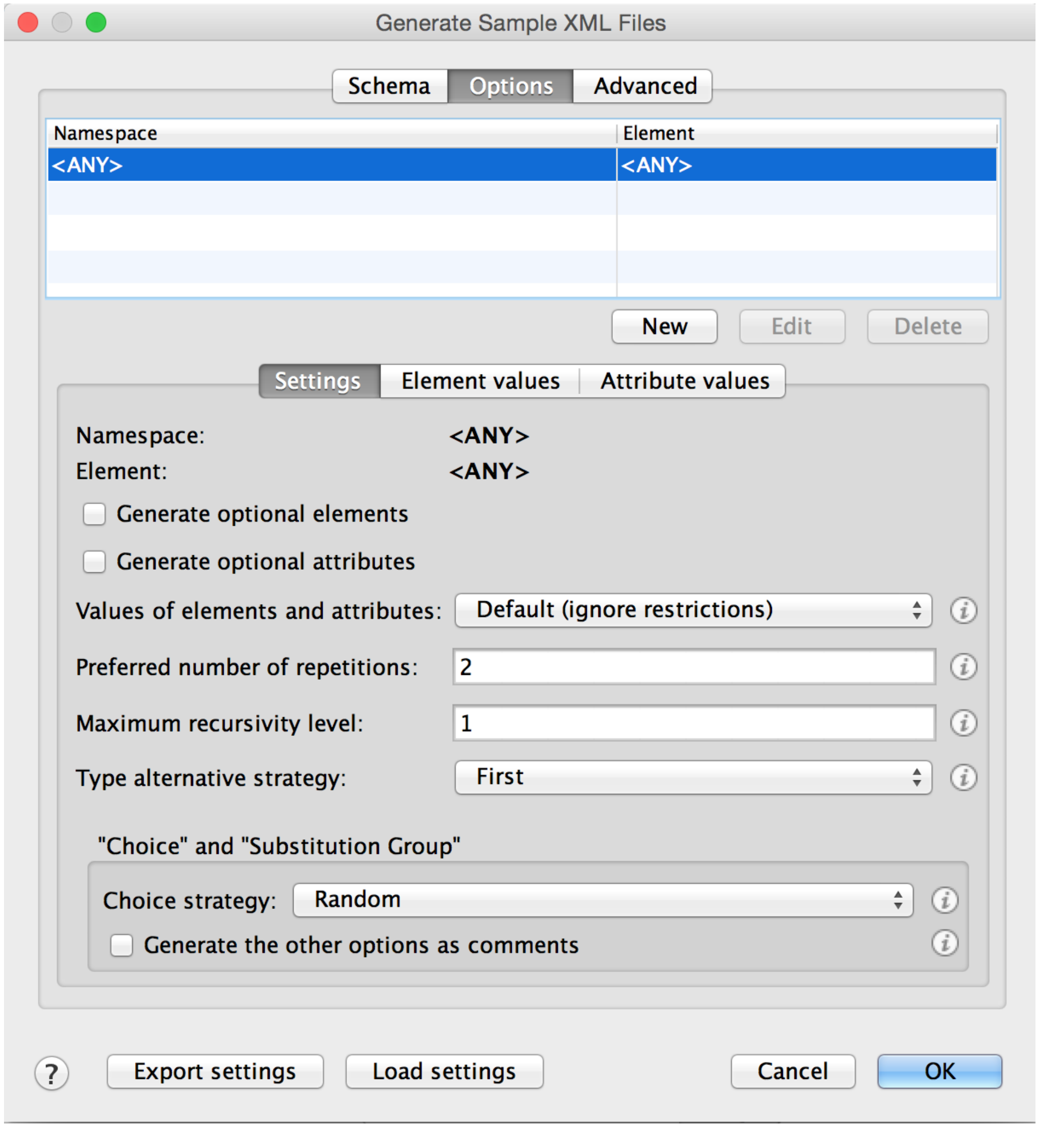
e) The “Root Element” box is also filled in with the default value “Ingest_LDD”. Use the down arrow and scroll bar to select the product type you want — for example, Product_Observational.

f) The “Output folder” box is filled in with the path to your common-schema file. Change this to the path to your working directry, if it is different.

g) In the “Filename prefix” box, enter the base file name for your label template file. As noted at the right, it will be given the extension *.xml.

Step 3: Select Options

a) **Do not select “OK”.** From the three tabs at the top, select the “Options” tab. You will get a window like the one shown in Figure D-4



Namespace	Element
<ANY>	<ANY>

Namespace: <ANY>
Element: <ANY>
☐ Generate optional elements
☐ Generate optional attributes
Values of elements and attributes: Default (ignore restrictions)
Preferred number of repetitions: 2
Maximum recursivity level: 1
Type alternative strategy: First
"Choice" and "Substitution Group"
Choice strategy: Random
☐ Generate the other options as comments

Figure D-4

b) Check the boxes for “Generate optional elements” and “Generate optional attributes”.

c) For the “Value of elements and attributes” you have three choices:

> None
> Default (ignore restrictions)
> Random (apply restrictions)

> You should select “None”; this means that all of the attributes in the label template will have empty values. The empty values will be flagged by the real time validation function of the oXygen XML editor; you can insert values and clear the error flags when you edit the template later.
>
> If you select “Default (ignore restrictions)”, meaningless values will be inserted. The real-time oXygen validation may not report errors. If you select “Random (apply restrictions)”, plausible values will be inserted. In both cases, you will have to check the template *very* carefully to prevent bad values from leaking into the final labels. It is simpler and safer to select “None” and work through the template, clearing flagged errors (empty values).

d) Set “Preferred number of repetitions” to “1”. If you need repetitions, you can add them later when you edit the template.

e) In the “Type alternative strategy” and “Choice strategy” boxes, select “First”. If you select “Random”, the results may be unpredictable. “Random” is useful if you’re simply interested in surveying example labels; when generating a label template, you want predictable results.

f) Check the “Generate the other options as comments”. This fleshes out the template with all possible options. Removing the “comment” notation is not difficult; but removing the large number of unwanted options is the primary disadvantage of this method.

Step 4: Create the label template (XML) document

a) Click ‘OK’ at the bottom right of the window.

b) The label template will auto-display in the XAE.

c) The label template has everything you need (and quite a bit more). Options included as comments will be set off by “<!--“ at the left margin (and by “-->” after the corresponding end tag, which is harder to spot). You can accept the ones you want by removing the special punctuation; you can remove the ones you don’t want by deleting the appropriate rows.

d) Save the label template file to your working directory; it will have the unique file name you specified on the Schema page (Figure D-3).

Step 5: Ensure the label template is valid (or not)

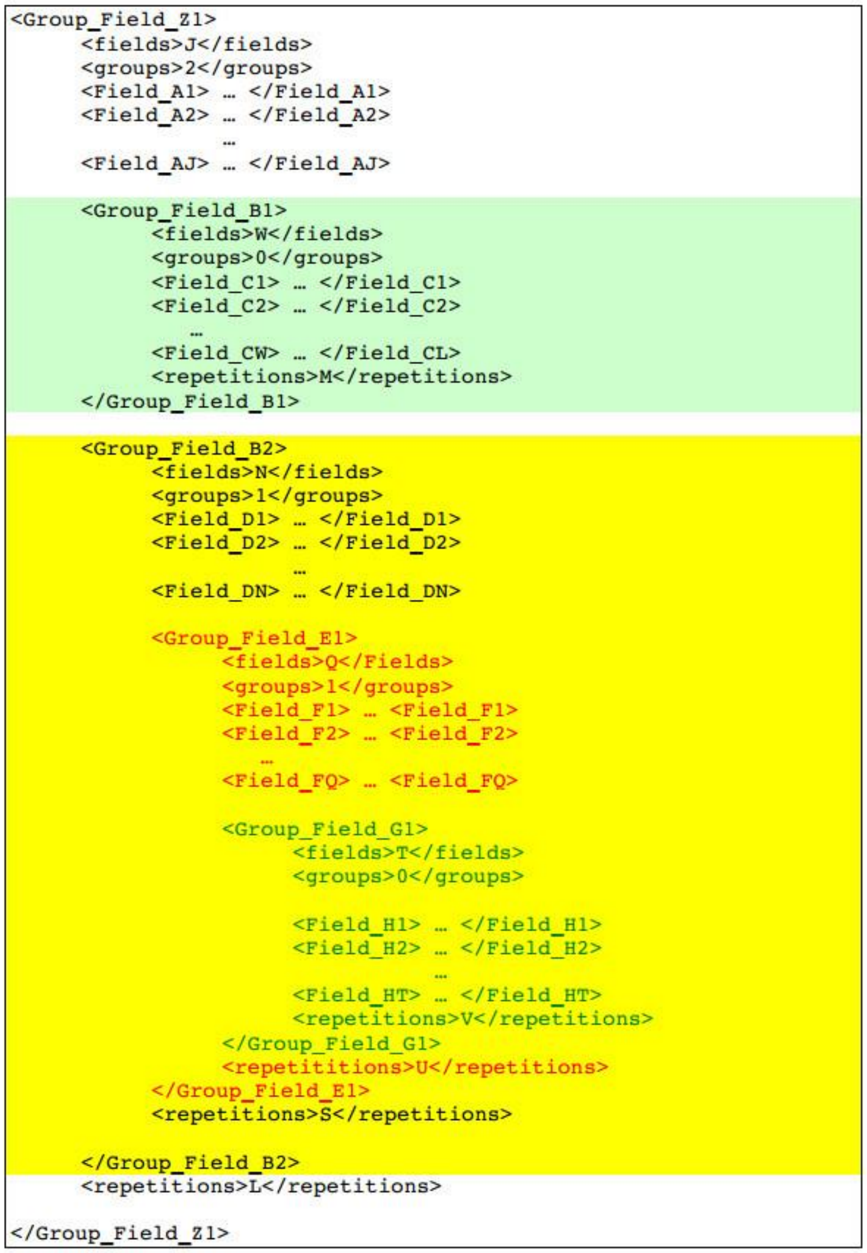
a) Your initial template is very likely not ‘valid’; little red boxes (or lines) in the right scroll bar indicate where errors exist. Most, if not all, of the errors in the initial template result from missing values.

b) As you proceed with editing the template (Section 7 of this document), you will clear the errors, and the red boxes will disappear. Note that the editor may not flag all of the errors initially. As you clear errors early in the file, red boxes will appear later in the file for errors that have been newly identified.

c) Once a full set of values has been specified, the label template should become valid, indicated by a green box at the top of the righthand scroll bar.

APPENDIX E FIELD AND GROUP COUNTING IN NESTED STRUCTURES

This appendix illustrates the steps that can be used to accurately identify the values (e.g., counts) for the <fields> and <groups> attributes in a nested structure; such as TABLE_CHARACTER or TABLE_BINARY.



```
<Group_Field_Z1>
     <fields>J</fields>
     <groups>2</groups>
     <Field_A1> … </Field_A1>
     <Field_A2> … </Field_A2>
                …
     <Field_AJ> … </Field_AJ>

     <Group_Field_B1>
          <fields>W</fields>
          <groups>0</groups>
          <Field_C1> … </Field_C1>
          <Field_C2> … </Field_C2>
             …
          <Field_CW> … </Field_CL>
          <repetitions>M</repetitions>
     </Group_Field_B1>

     <Group_Field_B2>
          <fields>N</fields>
          <groups>1</groups>
          <Field_D1> … </Field_D1>
          <Field_D2> … </Field_D2>
                     …
          <Field_DN> … </Field_DN>

          <Group_Field_E1>
               <fields>Q</Fields>
               <groups>1</groups>
               <Field_F1> … <Field_F1>
               <Field_F2> … <Field_F2>
                  …
               <Field_FQ> … <Field_FQ>

               <Group_Field_G1>
                    <fields>T</fields>
                    <groups>0</groups>

                    <Field_H1> … </Field_H1>
                    <Field_H2> … </Field_H2>
                               …
                    <Field_HT> … </Field_HT>
                    <repetitions>V</repetitions>
               </Group_Field_G1>
               <repetititions>U</repetitions>
          </Group_Field_E1>
          <repetitions>S</repetitions>

     </Group_Field_B2>
     <repetitions>L</repetitions>

</Group_Field_Z1>
```

- Group_Field_Z1 has J fields plus B1 plus B2 repeated L times

= {J + B1 + B2}*L total fields
= {J + W*M + [N + (Q + T*V)*U]*S}*L total fields

- Group_Field_B1 has W fields repeated M times
  = W*M total fields

- Group_Field_B2 has N fields plus E1 repeated S times
  = [N + E1]*S total fields
  = [N + (Q + T*V)*U]*S total fields

  - Group_Field_E1 has Q fields plus G1 repeated U times
    = (Q + G1)*U total fields
    = (Q + T*V)*U total fields

    - Group_Field_G1 has T fields repeated V times
      = T*V total fields

Note that the minimum value for <fields> is “0” and the sum <fields> + <groups> must be at least “1”.

APPENDIX F LID FORMATION RULES FOR PDS4 CONTEXT PRODUCTS

This appendix describes the formation rules used by PDS/EN to create unique identifiers for all PDS context products. These rules should be taken as 'guidelines' rather than requirements; depending on special circumstances, there may be occasional variants — such as for legacy context products.

A versioned identifier (LIDVID) is used to locate products within the PDS; every version of every product within PDS has a unique LIDVID. The steward for context products is the Engineering Node (EN). Data preparers should request LIDVIDs for new or updated context products from EN.

LIDVIDs for context products are formed as follows, where the composite key formed by [1] through [3] uniquely identifies the PDS4 product:

urn:nasa:pds:context:[1]:[2].[3]::[version_id]

Field [1] is dependent on the type of context product; Field [2] is often a subtype of Field [1] or the Field [1] value repeated. See notes (n) below for details where table space is limited.

Value of Attribute <product_data_object> in Product_Context Class	**Field [1] Value**	**Field [2] (select one from the choices offered, or see Notes below)**	**Field [3] (select one from the choices offered, or see Notes below)**
Agency	agency	agency	esa nasa
Facility (1)	facility	laboratory observatory	(A)
Instrument	instrument	(2)	(B)
Instrument_Host	instrument_host	(3)	(C)
Investigation	investigation	campaign individual mission other	(D)
Node	node	node	(E)
Other	(4)	(4)	(4)
PDS_Affiliate	personnel	personnel	(F)
PDS_Guest	personnel	personnel	(F)
Resource	resource	resource	(G)
Target	target	(5)	(H)
Telescope	telescope	(6)	(I)

Notes:

Follow character constraints listed below, such as converting to lower case.

(1) The Facility LID construction is not to be used for spacecraft; use Instrument_Host instead.
(2) Chose one of the following:

    facility.<instID>
    insthost.<instID>
    investigation.<instID>
    person.<instID>
    telescope.<instID>
    <instID>

    where <instID> is a recognized identifier for the instrument. It may be used alone (sixth choice) only for spacecraft instruments.
(3) Select a value from the enumerated value list for Instrument_Host.type, following constraints, such as for “lander” and “rover”.
(4) Not used
(5) Select a value from the enumerated value list for Target.type, such as “comet” or “star”.
(6) Enter the name of, or an identifier for, the parent organization, owner, or manager of the telescope (the entity that named the telescope) — a character string that would be easily recognized and accepted. For example, and depending on the type of ‘owner’, this could be (A) or (F) below.

(A) Enter an identifier for the facility such as kpno, lowell, or mcdonald
(B) Enter an identifier for the facility, instrument host, investigation, person, or telescope, as appropriate.
(C) Enter an identifier for the instrument host
(D) Enter the name of or an identifier for the investigation
(E) Enter the preferred acronym for the node, subnode, or data node.
(F) Enter the name of the individual in the format givennameorinitial_familyname
(G) Enter an identifier for the resource
(H) Enter the name of the target
(I) Enter an identifier for the telescope

Character Constraints:

Character strings for Fields [1], [2], and [3] must adhere to the following constraints:

```
(1) lower case letters    a-z   ASCII 0x61 to 0x7a
(2) digits                0-9   ASCII 0x30 to 0x39
```

```
(3) dash              "-"   ASCII 0x2D,
(4) period            "."   ASCII 0x2E, and
(5) underscore        "_"   ASCII 0x5F
```

Recommendations to ensure conformity in conversions from other contexts:

```
(1) replace space with underscore (i.e., replace " " with "_")
(2) replace forward slash with dash (i.e., replace "/" with "-")
(3) replace left parenthesis with NULL (i.e., replace "(" with "")
(4) replace right parenthesis with NULL (i.e., replace ")" with "")
(5) replace ampersand with dash (i.e., replace "&" with "-")
```

Examples:

1. agency:
   – European Space Agency:
     - `urn:nasa:pds:context:agency:agency.esa`
   – National Aeronautics and Space Administration:
     - `urn:nasa:pds:context:agency:agency.nasa`

2. facility:
   – Hofmeister Laboratory at Washington University:
     - `urn:nasa:pds:context:facility:laboratory.hofmeister`
   – Kitt Peak National Observatory:
     - `urn:nasa:pds:context:facility:observatory.kpno`
   – McDonald Observatory:
     - `urn:nasa:pds:context:facility:observatory.mcdonald`

3. instrument:
   – Grundy *et al*. Nicolet Nexus 670 FTIR Spectrometer:
     - `urn:nasa:pds:context:instrument:facility.i2041.lab9884`
   – Voyager 1 Narrow Angle Imaging Science Subsystem:
     - `urn:nasa:pds:context:instrument:insthost.issn.vg1`
   – Voyager 2 Narrow Angle Imaging Science Subsystem:
     - `urn:nasa:pds:context:instrument:insthost.issn.vg2`
   – Cassini Orbiter Narrow Angle Imaging Science Subsystem:
     - `urn:nasa:pds:context:instrument:insthost.issna.co`
   – CCD Imager with Tektronik 2000x2000 chip on Kitt Peak 2.1m Telescope
     - `urn:nasa:pds:context:instrument:telescope.cfim_t2ka.kpno2m1`
   – Imaging Grism Instrument at McDonald Observatory (movable among telescopes):
     - `urn:nasa:pds:context:instrument:facility.igi.mcdonald`

4. instrument_host:
   – European Southern Observatory 2.2 meter telescope:
     - `urn:nasa:pds:context:instrument_host:earth-based.eso_2m2`
   – Voyager 1 spacecraft:
     - `urn:nasa:pds:context:instrument_host:spacecraft.vg1`

5. investigation:
   – Eric Gurrola Ph. D. dissertation:

  - `urn:nasa:pds:context:investigation:individual.gurrola_phd_1995`
- Literature search by A. B. Smith:
  - `urn:nasa:pds:context:investigation:`
    `individual.ab_smith_literature_search`
- Voyager 1 Mission:
  - `urn:nasa:pds:context:investigation:mission.voyager`
- Cassini-Huygens Mission:
  - `urn:nasa:pds:context:investigation:mission.cassini-huygens`
- Lowell Observatory Comet Data Base (many observations, various telescopes):
  - `urn:nasa:pds:context:investigation:campaign.lowell_comet_db`

6. node:
   - Atmospheres Node:
     - `urn:nasa:pds:context:node:node.atmos`
   - SBN PSI subnode:
     - `urn:nasa:pds:context:node:node.sbnpsi`

7. personnel:
   -- PDS personnel:
     - `urn:nasa:pds:context:personnel:personnel.a_culver`

8. resource:
   -- PDS resource:
     - `urn:nasa:pds:context:resource:resource.1_ceres__browsert_asteroids`

9. targets:
   - Amalthea (satellite):
     - `urn:nasa:pds:context:target:satellite.amalthea`
   - Amalthea (asteroid):
     - `urn:nasa:pds:context:target:asteroid.113_amalthea`
   - Alpha Lyr (star):
     - `urn:nasa:pds:context:target:star.alpha_lyr`

10. telescope:
    - Kitt Peak 2.1 meter telescope:
      - `urn:nasa:pds:context:telescope:kpno.2m1`
    - McDonald 2.1 meter Struve telescope:
      - `urn:nasa:pds:context:telescope:mcdonald.struve2m1`
    - Arecibo (National Astronomy and Ionosphere Center) 305 meter radio telescope:
      - `urn:nasa:pds:context:telescope:naic.305m`

APPENDIX G PROCESS FOR SELECTING A PRODUCT CLASS

This appendix describes the decision process used to begin the selection of appropriate product classes when the choice is not obvious. For simplicity this outline assumes there is one data object in the product. The outline is shown schematically in Figure G-1.

(1) Is the object digital, a bundle, or an update to an existing product? If none of these, use Product_Context.

(2) Does the object contain PDS operational data? If so, use one of several Operational_Product classes defined in the Information Model (these are not, strictly speaking, 'science' data).

(3) Is this an aggregation product? If so, choose between Product_Collection and Product_Bundle. Product_Collection contains an Inventory table listing basic products. Product_Bundle contains a list of member collections.

(4) Is this a product that 'supports' the archive rather than being data per se? That is …

(a) Is this a ZIP compressed file? If so, use Product_Zipped.

(b) Is this an XML schema and/or Schematron file? If so, use Product_XML_Schema.

(c) If (4) but neither (a) nor (b) and it is ASCII_Short_String_Collapsed with <CR><LF> record delimiters, then use Product_Text_File.

(d) If (4) but not (a), (b), or (c) work with the consulting DN to determine the appropriate product type.

(5) Is this a SPICE kernel? If so, use Product_SPICE_Kernel.

(6) Is this solely a 'finding aid' for humans? If so,

(a) Is this used in an image index? If so, use Product_Thumbnail.

(b) If not to be used in an image index, use Product_Browse.

(7) Is this intended primarily to be read by humans? If so, use Product_Document.

(8) Does this update an existing product? If so, use Product_Update.

(9) Does the digital object contain observational data?

(a) If it contains observational data, does it have one of the following PDS4 structures: Array, Encoded_Header, Header, Stream_Text, Table_Binary, Table_Character, or Table_Delimited?

(i) If it has one of these structures and the processing level is ‘raw’, use Product_Observational.
(ii) If it has one of these structures, is not ‘raw’, and its function is primarily to support other products in the archive, use Product_Ancillary.
(iii) If it has one of these structures, is not ‘raw’, and its function is not primarily to support other products in the archive, use Product_Observational.
(iv) If it does not have one of these structures but it is in an original format (such as from the observing system), use Product_Native (see Section 9E for special requirements)
(v) If it does not have one of these structures and is not in an original observing system format, use Product_Ancillary.

(b) If it does not contain observational data, use Product_Ancillary.

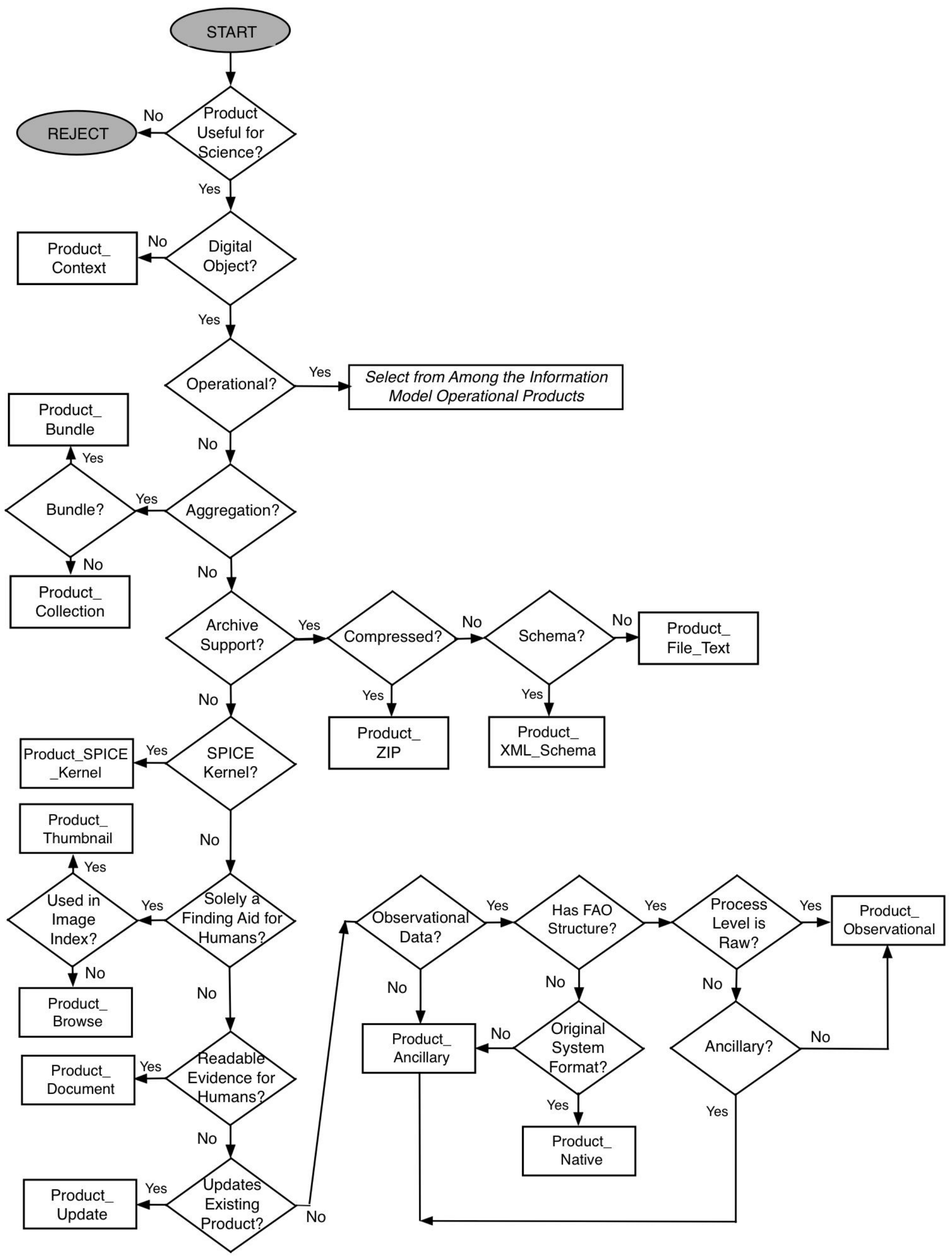

START
Product Useful for Science?
No
REJECT
Yes
Digital Object?
No
Product_ Context
Yes
Operational?
Yes
Select from Among the Information Model Operational Products
No
Aggregation?
Yes
Bundle?
Yes
Product_ Bundle
No
Product_ Collection
No
Archive Support?
Yes
Compressed?
Yes
Product_ ZIP
No
Schema?
Yes
Product_ XML_Schema
No
Product_ File_Text
No
SPICE Kernel?
Yes
Product_SPICE _Kernel
No
Solely a Finding Aid for Humans?
Yes
Used in Image Index?
Yes
Product_ Thumbnail
No
Product_ Browse
No
Readable Evidence for Humans?
Yes
Product_ Document
No
Updates Existing Product?
Yes
Product_ Update
No
Observational Data?
No
Product_ Ancillary
Yes
Has FAO Structure?
No
Original System Format?
No
Yes
Product_ Native
Yes
Process Level is Raw?
Yes
Product_ Observational
No
Ancillary?
No
Yes

Figure G-1

APPENDIX H PERMITTED SCHEMA MODIFICATIONS

The PDS4 common-schema (XSD) and Schematron file (SCH) may not be modified by missions and other data providers; however, they may saved and then edited as mission (or other) schemas with the approval of the PDS consulting node. This represents an alternative to the ‘template’ method for generating labels (Section 7). These ‘tailored’ schemas must be consistent with the common files in the sense that parameter constraints must be *at least as strict* as in the parent. The mission or other data provider can also add new classes and attributes to the tailored schema. They can then be used as input to label generating software.

The following is a summary of the types of modifications that you, as the data provider, might want to make in tailoring the common-schema:

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, but less than or equal to the value of maxOccurs in the master-common-schema.
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) The During the initial tailoring, the PDS supporting node staff may require common-schema classes and attributes that are identified as optional in the common-schema.
g) The PDS supporting node may require the use of specific discipline schema classes and attributes. These will appear in the additional classes in the Mission_Area and / or the Discipline_Area -- subsections within the Observation_Area.
h) The data provider may insert additional specific mission schema classes and attributes. These will appear 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.

Additional constraints can be addressed imposed by creating new and/or modifying existing “rules” in a Schematron file (SCH) that is discipline or mission specific. For additional information, consult your DN.