Creating the Ingest LDD Dictionary Input File

Prerequisites

If you have not already gotten familiar with a validating editor, or at least a schema-aware editor, you should take the time to do so first. You will thank me later.

If you have not tried to write or read a PDS4 XML label before, you should probably at least walk through one with a knowledgeable guide. Contact your PDS node consultant.

If you are using a schema-aware or validating editor, you should know how to create a new XML file from a schema file using your editor, and how to reference the schema and the related Schematron file if you are validating using your editor. You should use the PDS4 namespace schema to create your XML file, and select the **Ingest_LDD** element as your root element.

If you are using a plain text editor, you should know how to create a new XML file from scratch with all the appropriate schema references, or have a good PDS4 template to follow. Your root element - the one containing the XSD schema references - is called **Ingest_LDD**. In this case you are also going to need some way to validate the result before you try to run it through *LDDTool*. Because the *Ingest_LDD* structure is part of the PDS4 Information Model, you can use the PDS4 *Validate Tool* to do this.

Some additional useful links:

- You may find useful context information from Using Local Dictionaries.
- File:LDDtemplate 1900.zip contains a minimal skeleton template to use as a starting point, if you need one (you'll still have to adapt it for your schema locations).
- File:LDDTool 1900 examples.zip contains a working set of input *Ingest_LDD* files and labels which use the classes defined to illustrate the dictionary techniques described in the pages referenced below.

Basic Strategies

The **Ingest_LDD** class is the root of the XML document you are creating. In it, you are going to be defining a series of attributes and classes. You can think of attributes as scalars - single-valued entities that can be used to build classes. Classes can be nested, so one class can contain other classes as well as attributes. Note that classes cannot be nested recursively - either directly or indirectly.

Following are some emerging best practices we've noted at SBN.

Organization of the File Contents

All attributes must be defined first; classes must be defined afterward. Within each section, however, there is no enforced ordering. In fact, it is not necessary to define one class before including it as a component of another class - although it has to be defined in the file somewhere.

For non-trivial dictionaries you should give some thought to organizing the definitions in each section to facilitate development and maintenance. Attributes can be sorted alphabetically, for example, or grouped by purpose, subsystem, or some other mission-specific/whimsical criterion. You may also find it convenient to organize your class definitions in some sort of hierarchy - defining lowest level subclasses first, for example, then defining each successive level of containing classes.

Naming Attributes and Classes

The PDS and discipline namespaces employ a simple naming convention with these features:

- Words in the name are separated by underscores.
- Words in attribute names are all lower-case.
- Words in class names are capitalized.
- Abbreviations are generally avoided.
- Word order usually follows English grammar unless there's a good reason not to.

There are rare exceptions, but on the whole these are followed fairly consistently in the shared namespaces. There is no requirement that you use these conventions in your attribute names and classes, but you might consider the aesthetics and readability of the resulting label if you use conventions that are very far removed from the above.

You are constrained to ASCII characters, so don't go nuts.

Grouping Attributes into Classes

In general, you can group attributes into classes and subclasses, or not, any way you want. The functional groupings established by comments in most PDS3 labels are, in general, a reasonable approach to class definitions.

SBN recommends that all attributes be a member of some class from the dictionary. It may be valid to have "naked" attributes from mission dictionaries hanging around in the label `<Mission_Area>`, but from a user's point of view these attributes usually are lacking something in context.

Beware of Slacking Off

You *must* provide real, substantive definitions for all your defined attributes, and should endeavor to do so for all your classes. These definitions are part of the external peer review - reviewers will judge them. You do not want to be found wanting in this respect.

If your attribute has a unit of measure, you ***must*** include the `<unit_of_measure_type>` in the definition. It is ***not*** sufficient to say, for example, "measured in nm" in the definition text without including `unit_of_measure_type` in the attribute definition.

Use *TBD* sparingly, if at all. For non-trivial dictionaries, letting the population of *TBD*s grow unchecked can lead to major issues just before - or worse, at - review.

A Note About Complexity

The Filling Out the Ingest_LDD Class topic, below, only describes the basic techniques available in the **Ingest_LDD** class. It is possible to define fairly complex relationships within this dictionary and to other namespaces. In general, though, SBN discourages this for mission dictionaries. Internal complexity often leads to confusion on the part of users not affiliated with the original team, and referencing other namespaces establishes a connection between dictionaries that are under the control of two or more different stewards who may or may not be aware of any underlying assumptions that would be invalidated by future development, especially after end of mission. So on the whole, try hard to keep it simple and localized. If that seems impossible for your metadata or mission organization, check in with your PDS node consultant for examples of more complex procedures that haven't ripped asunder the fabric of space-time.

Filling Out the Ingest_LDD Class

Bear in mind while reading these descriptions that *LDDTool* and <*Ingest_LDD*>, as its input template, are products that will be updated regularly to keep pace with the development of the PDS4 information model and the needs of the discipline dictionaries in development. The following sections describe the stable elements that provide the basic capabilities used in all dictionaries. You will come across undocumented options below and in the PDS master schema that are not yet ready even for this level of exposure. If you think you need them or could put them to good use, contact your PDS node consultant for more information.

We'll tackle this one section at a time, starting with the attributes at the top of the <*Ingest_LDD*> class itself and moving down through the file structure:

1. **Dictionary Parameters** - These attributes at the beginning of the *Ingest_LDD* file define the dictionary name, namespace abbreviation, steward, and so on.
   Filling Out the Ingest_LDD Attributes
2. **Attribute Definitions** - This section will contain one *DD_Attribute* class for every attribute in the dictionary. All attributes must be defined in this section before any classes may be defined.
   Filling Out the DD_Attribute Class
3. **Class Definitions** - This section will contain one *DD_Class* class for every class to be included in the dictionary. All classes must be defined in this section before any Schematron rules may be defined.
   Filling Out the DD_Class Class

And for the brave-hearted, here are some Advanced Ingest_LDD/LDDTool Techniques**:**

- **Schematron Rules** - Following your class definitions, you can add *DD_Rule* classes to define Schematron rules to be included in the output Schematron file. The *LDDTool* will automatically generate the Schematron rules required to support enumerated value lists, so you'll only need this if you want to specify additional constraints beyond those provided by the *DD_Attribute* and *DD_Class* definitions.
  Filling Out the DD_Rule Class
- **Referencing elements of other namespaces** - Sometimes a dictionary writer may need to incorporate classes from another namespace into classes in the dictionary being written. This can be accomplished via a syntactical trick that should be used with caution.
  Advanced LDDTool Techniques: Cross-referencing Namespace Elements
- ***Choice* Lists** - There is a *xs:choice* construct in the XML Schema Definition language that lets you specify that one (or more) of several different elements can be included in a containing complex element. *Choice* lists that allow more than one element to be included also circumvent the usual strict ordering of the XML Schema Definition language.
  Advanced LDDTool Techniques**:** *Choice* Lists
- ***Any* Blocks** - The XML Schema Definition language *any* construct is designed specifically to stop schema validation at a particular point in the class definition, after which you could, technically, include any valid XML and a schema-based validator would let it pass unremarked. You should *never* do this in a mission dictionary unless directed to do so by a knowledgeable PDS dictionary expert.
  Advanced LDDTool Techniques: *Any* Blocks