Schema Referencing in PDS4 Labels

There are several ways to tie schema files to the XML documents they define in order to validate the documents and take advantage of schema-aware editors; but in general, the methods used to locate schema files are not compatible with each other. In other words, XML editors need to pick the method they want to use, and then use it consistently. Trying to change methods generally involves changing software settings and/or editing the schema references in the XML files.

The PDS4 schema library is relatively complex and interlinked. That is, the PDS4 dictionary schemas - the ones that define the core PDS and discipline name spaces as well as the mission dictionaries - cross-reference each other. In order for any particular software environment, then, to be able to resolve all schema references reliably, it will be rather important that the same technique be used in all dictionary schemas and all label files, regardless of source. This must also be done in an environment-agnostic way, or you will have to edit schema files each time you try to run validation on a new machine, or even in a new directory in the same disk space.

This page describes how to set up your PDS4 labels to be consistent with the PDS schema library and remove environmental dependence from your schema references as far as possible. This method is strongly recommended to PDS4 data preparers. In fact, your node consultant may insist on it in order to have consistent and reliable validation of your deliveries.

Preliminaries

PDS-controlled namespaces will almost always be defined by a pair of related schema files: an XML Schema (*.xsd*) file to define the class structures and general data types; and a Schematron (*.sch*) file to define enumerated value lists and conditional structure relationships (e.g., you must use PDS attribute *A* or PDS attribute *B*, but not both). You will need to tie your labels to *both* of these files. The Schematron file will be referenced in the XML prolog; the XSD file will be referenced in the document root tag (<Product_Observational>, for example).

Note: Schema File vs. Namespace

URIs (Uniform Resource Identifiers) are used to identify both namespaces and the files that define those namespaces. While it is easy, given the notational conventions described below, to conflate these two things, they are and remain different concepts to your software. The namespace URI is a logical identifier - it refers to the concept of the dictionary, irrespective of minor version changes. That is, version 1.3 of the PDS core namespace, for example, has exactly the same URI as version 1.5 of the same namespace. (Version 2.0, though, would have a different URI.)

The schema URIs, however, must resolve to physical files. It is the schema URIs that control the version of the namespace actually applied to the label for editing assistance and for validation.

The practical upshot for PDS4 labels is that when you are referencing a schema file, your URI will contain a file name. When you are referencing a namespace, it will not. And in order to allow for reasonable transportability, file system references will be replaced by URI references that can be resolved through an XML Catalog file.

Schematron (SCH) References

Schematron references are placed in the prolog of the document following the XML declaration. Schematron files are referenced by xml-model processing instructions. (The *prolog* is everything before the document root tag; processing instructions are delimited by the character pairs **<?** and **?>** - same as for the XML declaration.)

The xml-model processing instruction is the focus of the W3C standard "Associating Schemas with XML Documents". It exists to provide an explicit link between an XML document and a schema that

defines its valid content. PDS uses the xml-model processing instruction to associate Schematron-type schema files, specifically, with a label. (The XSD schema files are associated through schemaLocation declarations.)

If your software (your editor, for example) has implemented the "Associating Schemas" standard, then you should use one of these two forms for xml-model in your PDS4 labels:

<?xml-model href="http://pds.nasa.gov/pds4/pds/v1/PDS4_PDS_1201.sch" schematypens="http://purl.oclc.org/dsdl/schematron"?>

if this is compatible with you processing environment, or:

<?xml-model href="http://pds.nasa.gov/pds4/pds/v1/PDS4_PDS_1201.sch"?>

if it is not. Here's what is going on:

href="http://pds.nasa.gov/pds4/pds/v1/PDS4_PDS_1201.sch"
href is required, and to be compliant with the "Associating Schemas" standard, it must be a URI that maps to a physical file. However, because the value of href is a URI, editors that implement the XML Catalog standard along with the schema association standard should use any relevant XML catalog entries to help resolve the href reference. You should keep that in mind when formulating your XML catalog entries. Also, note that the href value must be a single URI, so you will need to include one xml-model statement for every Schematron file you wish to associate with the label. Start each xml-model statement on a new line to avoid confusion and trouble down the line.
schematypens="http://purl.oclc.org/dsdl/schematron"
The optional schematypens attribute gives any software that cares to check a hint about what kind of schema it can expect to find when it resolves the href URI to a physical file. The namespace shown here is the official namespace URI for ISO Schematron - the version used in PDS4 dictionaries. There are other flavors of Schematron out there, and they are **not** compatible or interchangeable with ISO Schematron, so if you are working in an environment that has several different Schematron implementations available, it is important to set this value, and your options/preferences, for *ISO Schematron*. In the absence of schematypens, a robust processor would read the schema file to discover its type and act accordingly. Less robust environments might signal a warning if they determine the schema type is not what they expected, or fail if it is not something they can process.

> **Note for All Users:** You ***must*** provide a separate <?xml-model> statement for ***each*** Schematron file you want to use in validation. Including multiple *href* values in a single *xml-model* statement leads to undefined behavior.

> **Note for *Eclipse* Users:** The *Eclipse* editor and its Schematron plug-in have a couple of significant limitations:
>
> 1. The href value must be a physical file location relative to the label in the current disk space. Web references and URIs will not resolve, even with XML catalog file entries available, and absolute file references don't seem to work, either. This is a ***major*** drawback with *Eclipse* if you need schema references that are environment-independent.
> 2. The presence of a schematypens pseudo-attribute will be flagged as an error.

There are other optional pseudo-attributes for xml-model that are unlikely, at least as of this writing, to show up in PDS4 labels, but they do at least have a format definition in the "Associating Schemas" standard. The ones you are most likely to see include:

- **type:** The value should be a content-type descriptor like those you would find in an HTTP header.

- **charset:** The value specifies a character set using standard abbreviations like "US-ASCII" or "UTF-8".
- **title:** The value is the title of the schema document being referenced by href.

XML Schema (XSD) References

It is possible to reference *.xsd* files from various places in your label, but editing and debugging these references tends to be a lot easier when you've got them all in one place. So we recommend you put all your *.xsd* file references in a schemaLocation list inside the document root tag. For PDS4 labels, the document root tag will be one of the <Product_*> tags.

xsi:schemaLocation

The xsi:schemaLocation attribute that we will be using inside the document root tag belongs to the **XMLSchema-instance** namespace, which is in turn defined by the *XML Schema* standard. There are two elements of this name space you may encounter regularly in PDS labels: the xsi:schemaLocation in the document root tag, and the xsi:nil property that you may see or use in setting some label values to nil in particular circumstances.

Note that, in order to reference elements from the **XMLSchema-instance** namespace, you have to tell your validators that you plan to do so. You do this the same way you tell your validators about other namespaces (see "Namespace References", below). You don't usually have to provide a defining schema file reference for the **XMLSchema-instance** namespace in you xsi:schemaLocation list, though, because if your software implements that standard, then the definition will be coded into the system already.

To link to the relevant *.xsd* files, you assign a string value to xsi:schemaLocation. This string contains pairs of (*namespace URI*, *schema file URI*) strings (no parentheses or commas in the actual value). You can use blanks and line breaks freely within the value to keep things visually organized for yourself, fortunately. Here's a typical list of (namespace, schema) pairs from a prototype label developed for a Deep Impact spectral image observation:

```
xsi:schemaLocation=
  "http://pds.nasa.gov/pds4/pds/v1           http://pds.nasa.gov/pds4/pds/v1/PDS4_PDS_1201.xsd
   http://pds.nasa.gov/pds4/disp/v1          http://pds.nasa.gov/pds4/disp/v1/
PDS4_DISP_1100.xsd
   http://pds.nasa.gov/pds4/sp/v1            http://pds.nasa.gov/pds4/sp/v1/PDS4_SP_1100.xsd
   http://pds.nasa.gov/pds4/geom/v0          http://pds.nasa.gov/pds4/geom/v0/
PDS4_GEOM_0520.xsd
   http://pds.nasa.gov/pds4/sbn/v0           http://pds.nasa.gov/pds4/sbn/v0/sbnDD_0100.xsd
   http://pds.nasa.gov/pds4/mission/epoxi/v0  http://pds.nasa.gov/pds4/mission/epoxi/v0/
epoxiDD_0100.xsd"
```

The string in the first column, above, is the URI for the namespace. The string in the second column is a URI that will, with the help of either a web connection or an XML Catalog file, resolve to a physical file that can be loaded into the editor or validator. The first pair refers to the core PDS namespace. Here's what is going on:

http://pds.nasa.gov/pds4/pds/v1
This is the URI of version 1 of the PDS core namespace. This string will be the same in *every* PDS4 label you see until there's a Version 2.0.0.0 of the Information Model.
http://pds.nasa.gov/pds4/pds/v1/PDS4_PDS_1201.xsd

This reference points to the file that contains the definition of the core PDS namespace that will be used for the elements in this label. These sorts of URIs for PDS4 schema files will resolve to a physical file if you simply reference it via the HTTP protocol. In other words, if you put the string "http://pds.nasa.gov/pds4/pds/v1/PDS4_PDS_1201.xsd" into your web browser, you will see the schema file displayed. That is *not* a requirement of the URI standard, but rather a convenience that PDS has chosen to implement for its PDS4 schema collection.

Some things to note for your `xsi:schemaLocation` list:

- The PDS core namespace definition should come first. This is because the other PDS-controlled namespaces reference the PDS core name space, but might be referencing a different version of it - most likely an earlier version. You cannot have two simultaneous definitions of the same namespace in force at the same time, and if your software encounters multiple definitions it will typically take the first and warn you about later ones (some programs will let you change that behavior through preferences, so you should check for that option when you first configure a new editor or validator). You should *not* depend on discipline dictionaries loading the version of the core schema you want to use. For a start, it makes it difficult to get the right value into the required `<information_model_version>` element.
- You *should* reference all the discipline namespace dictionaries you are using in the label in your `xsi:schemaLocation` list. The file name for these PDS-controlled namespaces contains the encoded version number, which as of this writing is the only way to indicate which version of a discipline dictionary supports the structures in the label. For discipline and mission dictionaries, the differences between versions can be significant and incompatible.
- You *should* use the HTTP-style URI for the file references shown above for your schema files wherever possible, because these references can be easily trapped and resolved by simple XML Catalog file entries. This, in turn, makes it possible to validate the same label in different environments without having to change anything in the label itself, and that tends to make life easier for everyone involved in designing, editing, and validating labels. (Note that while *Eclipse* users can make use of this method for XML Schema files, they will *not* have this luxury with their Schematron references until someone writes a better plug-in.)

Finally, note that you *do* have to put all the schema location information in the same string; having more than one `xsi:schemaLocation` in your root tag is an error.

Namespace References

Every tag in your label must be associated with exactly one, specific namespace. You may, if you like, designate one namespace as the default namespace for your tags. If you designate the PDS core namespace as your default, for example, then when you want to use tags from another namespace, like a mission or discipline dictionary, you will have to identify that new namespace. There are several methods for doing this, all of which should yield the same result for validation purposes (unlike methods for referencing physical schema files described above).

Here we will describe two common methods that seem to be both simple and human reader-friendly for use in PDS4 labels.

Namespace Abbreviation Prefixes

In your document root tag, you can declare which namespaces you plan to reference and assign each a unique abbreviation via `xmlns` assignments. Here is a set of assignments that would correspond to the `xsi:schemaLocation` namespaces of the previous section:

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

```
    xmlns:sp   = "http://pds.nasa.gov/pds4/sp/v1"
    xmlns:geom = "http://pds.nasa.gov/pds4/geom/v0"
    xmlns:sbn  = "http://pds.nasa.gov/pds4/sbn/v0"
    xmlns:epoxi= "http://pds.nasa.gov/pds4/mission/epoxi/v0"
    xmlns:xsi  = "http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="...">
    ...
  </Product_Observational>
```

Here's what is going on:

xmlns = "http://pds.nasa.gov/pds4/pds/v1"
This defines the default namespace for the label as the PDS4 core namespace. Any tag without a prefix that doesn't declare a new default namespace (as described below) must be defined by the PDS4 core schema, which is in the xsi:schemaLocation list, of course. If you wanted to define a prefix for this namespace as well, you could - by using the syntax described next. You are not required to have a default namespace.
xmlns:disp = "http://pds.nasa.gov/pds4/disp/v1"
This associates the abbreviation **disp** with the Display Discipline Dictionary namespace. Now if you want to use an element from that dictionary, you can prefix the tag name with "disp:". So, for example, the <Display_Direction> class would look like this:

```
  <disp:Display_Direction>
    <disp:horizontal_display_axis>Sample</disp:horizontal_display_axis>
    <disp:horizontal_display_direction>Left ot Right</disp:horizontal_display_direction>
    <disp:vertical_display_axis>Line</disp:vertical_display_axis>
    <disp:vertical_display_direction>Bottom to Top</disp:vertical_display_direction>
  </disp:Display_Direction>
```

Couple of things to note:

- You *must* specify the namespace prefix for *all* tags that come from that namespace, even if they are inside other tags that already have the namespace prefix attached.
- You *must* specify the namespace prefix on both opening and closing tags. XML is a real stickler that way.
- There *must* be a one-to-one correspondence between namespaces and abbreviations (including the null abbreviation for the default namespace, if you have one). You cannot assign "disp" as the abbreviation for two different namespaces, nor can you define both "disp" and "d" to both be abbreviations for the Display Dictionary namespace.
- If the PDS core namespace is **not** the default namespace for your label, you **must** include the PDS core namespace prefix on your document root tag as well as all other tags from the core namespace. So, if your <Product_Observational> tag contains this:

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

then your document root tag should be "pds:Product_Observational", not just "Product_Observational"

One final note: The abbreviation you designate to use for each namespace is, technically, at your discretion. You can, as far as the XML standards are concerned, use totally non-standard abbreviations for *XMLSchema-instance*, for the PDS4 core namespace, or for any or all of the

dictionary namespaces you reference. *But you shouldn't*. The standard abbreviations for PDS-controlled namespaces are reserved by PDS permanently to be available as unique abbreviations that will help users immediately recognize label content and provenance. Deliberately undermining these associations with arbitrary namespace abbreviations is not a good thing for archive stability. Expect that PDS nodes and reviewers will take a dim view of that sort of thing.

Changing the Default Namespace

If prefixing tag names with namespace abbreviations annoys you, you can change the default namespace at any point in your label by including an xmlns assignment inside any tag. The new default will last until you close that tag, and you can change default namespaces again in nested tags if you like.

As an example, let's assume that we have defined the PDS core namespace as our default namespace in our <Product_Observational> document root, as in the previous examples, but have not defined any namespace abbreviations for things like the Display Discipline Dictionary. Now if we want to use the <Display_Direction> class, we need to change the default namespace to that of the Display Dictionary. We do that with an xmlns assignment within the <Display_Direction> tag:

```
<Display_Direction xmlns="http://pds.nasa.gov/pds4/disp/v1">
  <horizontal_display_axis>Sample</horizontal_display_axis>
  <horizontal_display_direction>Left ot Right</horizontal_display_direction>
  <vertical_display_axis>Line</vertical_display_axis>
  <vertical_display_direction>Bottom to Top</vertical_display_direction>
</Display_Direction>
```

The new default expires at the corresponding </Display_Direction> tag, and the default namespace reverts to whatever it was before the opening tag.

The usual couple of notes:

- The tag containing the xmlns assignment *must* be defined in the given namespace. So this, for example:

```
<Mission_Area xmlns="http://pds.nasa.gov/pds4/mission/epoxi/v0">
```

will fail validation because the Mission_Area tag allowed at that point in a PDS4 label is defined in the PDS core namespace, not the EPOXI mission namespace. In order to change the default namespace to the mission namespace, you must do so from a tag defined in the EPOXI mission dictionary. (This is where wrapper classes earn their keep.)

- You still need the xsi:schemaLocation assignment to tell software where to find the namespace definition. And it should be at the top of the file rather than here, to make it easier to debug validation issues and to better document the label content in a simple, consistent manner.
- Using this method you may find yourself typing the same namespace reference several times in a single file. This introduces an opportunity for subtle error if and when there is more than one major version of a namespace defined. It will be awhile before this happens for PDS-controlled namespaces, but missions may be more liberal with their major version changes. In general, you should not be referencing multiple major versions of the same namespace - but XML validators are unlikely to be able to catch this condition and flag it for you. Type carefully if you are working with a dictionary that has multiple major versions, and **never** include a xsi:schemaLocation for more than one major version of a namespace directly in you document root tag.

Abbreviations vs. Changing Defaults

As far as reasonably modern software is concerned, these two methods for identifying namespaces are completely equivalent. XML parsers should return the same result regardless of how the namespace for any given tag is identified. So which method you prefer will depend on things like personal aesthetics or which is easier to use with your schema-aware editor.

This equivalence extends to the parsers involved in the Schematron validation step. Both of these methods work equally well in schema-aware editors that support Schematron validation.

You should avoid using both methods in the same label, though. As far as XML standards and parsers are concerned, you can - but it is visually and logically confusing for users. So try not to.