Understanding XML Catalog Files

XML catalog files use some terminology that can be fairly opaque to those new to XML. Following is an explanation of the key terms used in the XML Catalog standard and their relevance to the PDS4 context.

Identifiers: Public vs. System

The concepts of *public identifier* and *system identifier* predate XML. Both concepts were key in the pre-OASIS world of SGML (the ancestor of XML). These identifiers allowed a document author to reference a file external to his own document. Typically, this would be a Document Type Definition (DTD). DTDs predate schemas, but do the same sort of job - defining the valid content of an SGML file. Standard DTDs were developed to provide interoperability between systems. Perhaps the most widely-known DTD is the DTD that defines the DocBook documentation system.

At this stage of the game, the distinction between *public* and *system* identifiers was clear and simple: The *public identifier* was a globally unique, permanent and invariant identifier assigned to a resource, like the DocBook DTD. The format of the *public identifier* was defined as part of the ISO 8879 (SGML) standard as the Formal Public Identifiers (FPIs) format, and there was (presumably still is) at least one registration authority to assign namespaces to insure that unique FPIs can be formulated by diverse organizations. The *public identifier* was clearly a logical identification of a resource.

In this regime, the *system identifier* was always a physical location - a reference to a file on disk, for example. The SGML standard required that at least one of the two identifiers was present, but did not require both.

Enter Catalog Files

At this point, the *public identifier* was a logical reference that could not easily be resolved, but at least it was transportable, unlike the *system identifier*. To address this problem, the SGML Open project, which eventually became OASIS, developed the first catalog-type standard (OASIS Technical Resolution 9401:1997) to map *public identifiers* to *system identifiers* in an external ("catalog") file, which could be referenced by applications.

Now, in this pre-XML world, this was a pretty straightforward task. The *public identifier* was always a logical reference, and the *system identifier* was always a physical reference to a locally accessible file. So a DocBook author, for example, could include both types of identifier in his source files as he was preparing them, and when he sent them out into the world the receivers could set their applications to ignore the system identifiers in the document and instead translate the *public identifiers* using their own catalog files. In other words, the application could choose whether the *public* or *system* identifiers should be "preferred" - a term that will come back later with much reduced significance for XML.

Time Passes...

SGML begat XML, the SGML Open group became OASIS Open, and URIs have largely supplanted FPIs. In XML documents, the *public identifier* is optional, while the *system identifier* is usually required (to identify things like name spaces and import files). But in XML, these references are also required to be URIs, which are themselves logical pointers. In the XML regime, the system identifier **does not** point to a physical location.

OK, it *might* point to a physical location - some URIs do. But in general URIs are **not required to be resolvable in themselves**, so you can't count on someone else's URI being directly resolvable to a physical file. Which is why XML documents may include **schemaLocation** attributes - to indicate the physical location of the files needed to define name spaces or to be imported into the current document.

XML Catalog Standard

So OASIS rolled up its sleeves and beefed up the early mapping standard to become the XML Catalog 1.0 standard, to address both SGML and XML mapping needs. The catalog file maps the values of *public identifiers*, *system identifiers* and URIs generally to (other) URIs that actually do resolve to a physical file. It will do this for anything your application considers to be an external id (either a *public identifier* or a *system identifier*), as well as for any other URIs it encounters. A few things to keep in mind when reading/writing catalog entries:

- The XML Catalog standard explicitly states that the first matching line is the one applied - anything else will be ignored. When you are writing your translation elements, put the most specific matches first, and the more general matches later. For example, if you are trying to match a URI that ends in a file name, put that element **before** any element that matches just the path.
- Applications can choose to be picky about URI formatting in your catalog files. According to the XML Catalog standard, catalog processors must normalize URIs before running a comparison, but some processors may be more liberal in what they'll recognize and translate for you than others if, for example, you use local OS path syntax rather than the Unix-like syntax technically required by the "file:" protocol. The oXygen editor, for example, is fairly lenient about URI formatting in the catalog file. Other applications may not be so forgiving.
- One of the consequences of the evolution from DTD and external (public/system) identifiers to XML and URIs is that the distinction between public and system identifiers is largely moot. The external identifiers in our PDS XML documents - the references to the XML Schema and XML Schema-Instance name spaces, for example - are not required to have system identifiers (the definitions are "built-in", as it were). Since everything else falls under the "URI" rubric, our XML Catalog files tend to contain only URI-type mappings.
- As a result, applications may be lenient about discriminating between public/system identifiers and general URIs when matching strings and applying mappings. For some applications, using a *system identifier* mapping rather than a URI mapping will still translate all occurrences of the matching URI, even if it technically isn't being used as a *system identifier*.
- It is possible to write complex catalog files, with elements for including additional files or branching from one catalog file to another. Most PDS data preparers and users don't need any of those complications. The standard set-up and a few simple URI mapping parameters will do the job for most of us.
- Catalog files are not transportable. They are the epitome of environment-specific configuration. So when following someone else's example, be particularly careful about the file specification URIs you will be translating to - they will depend critically on your local file system.

XML Catalog File Elements

Here is what you need to know to write or edit an XML Catalog file.

Every catalog file will begin with the usual <?xml> tag and possibly a <!DOCTYPE> declaration (some applications require it, some forbid it), followed by the <catalog> tag which begins the catalog information proper and identifies the namespace associated with the XML Catalog standard. These can be copied verbatim from any valid catalog file; if you use an XML Catalog generation tool, these

will be provided for you. The <catalog> tag may have a prefer attribute with a value of either "public" or "system". As explained above, for PDS purposes this preference setting is meaningless – we will only be mapping URIs, not external identifiers.

Between the <catalog> and </catalog> tags, these are the tags that will likely be most useful and most common in catalog files supporting PDS labels:

<uri name="*name_string*" uri="*physical_reference*"/>

The <uri> element does a straight one-to-one mapping from the URI given as the value of "name" to the URI given as the value of "uri". So *name_string* is what appears in the XML file, and *physical_reference* is the actual location of the file that contains the answer (the namespace definition, the XML fragment to be included, etc.). This **must** be resolvable. For most of our users this will resolve to a file on the local file system, so it will begin with the string "file:///". It could also resolve to a web location if that's the way you roll, in which case it will likely begin with something like "http:" or "ftp:". The URIs should both be URI-encoded, for safety.

<rewriteURI uriStartString="*old_prefix*" rewritePrefix="*new_prefix*"/>

The <rewriteURI> element can be used to map many URIs at once, based on a common initial substring in those URIs. For example, say you have reproduced the PDS schema directories in a local repository. You could then map all your PDS namespace references at once by replacing the "http://pds.nasa.gov/pds4" part of every namespace URI with a reference to the root directory of your schema repository. As with the <uri> element, the URI created must be resolvable. *Old_prefix* is the prefix as it appears in the XML file; *new_prefix* is the replacement that turns that string into a resolvable reference.

<uriSuffix uriSuffix="*uri_suffix*" uri="*physical_reference*"/>

The <uriSuffix> element matches based on the *end* of the URI string - so if the URI in the XML document ends in *uri_suffix*, then the entire URI is mapped to the *physical_reference* (which must, of course, be resolvable). Note that this is not at all like <rewriteURI>, which effectively does a string substitution on the URI from the XML document. <uriSuffix> matches based on the suffix only, but then expects to map this to a complete, new URI. (One of the few differences between the XML Catalog 1.0 and 1.1 standards is the addition of this element in the 1.1 standard.)

<delegateURI uriStartString="*prefix_string*" catalog="*physical_reference*"/>

The <delegateURI> element lets you hand off URI translation for a set of URIs to a different catalog file. This can be useful if you are working in a fairly complex environment where some of your URI translations are stable and some aren't (or some are in production mode and others in development). This could also be used to set up a hierarchy of public and private XML catalogs. When a URI in the XML document starts with the *prefix_string*, the URI will be immediately handed off to the catalog file indicated by the *physical_reference* for processing. (Note, though, that the catalog processing will stop at the *first match encountered*, so take care with where you locate your delegate element.)

There are analogous elements to the above for mapping *public identifiers* and *system identifiers*, as well as a <group> element for providing default preferences and base URIs for these elements, and a <nextCatalog> element for explicitly passing control to another catalog file (rather than letting your application work through a predefined list). In addition, all the elements listed above will take an xml:base attribute to specify a base URI, so that relative URIs can be turned into absolute URIs. For most PDS uses, where all required URIs are also required to be absolute and the public/system preference is not applicable, these are not necessary. If you think you might need or want them, read the standard carefully and have at it.

Some Simple Examples

Following are some simple XML Catalog files for a couple of common scenarios. Note that these all contain both the *DOCTYPE* reference and the catalog namespace reference ("urn:oasis:names:tc:entity:xmlns:xml:catalog"). Some PDS4 tools may choke on the *DOCTYPE* directive; it can be removed as long as the namespace reference remains in the `<catalog>` statement.

rewriteURI

This catalog file uses a single `rewriteURI` statement to map all PDS4 namespace schema references to a copy of the schema tree on a local (NFS-mounted) directory:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE catalog PUBLIC "-//OASIS//DTD XML Catalogs V1.1//EN" "http://www.oasis-
open.org/committees/entity/release/1.1/catalog.dtd">
<catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog">
  <rewriteURI uriStartString="http://pds.nasa.gov/pds4"
              rewritePrefix="file:///n/sbnops/lcltools/schema"/>
</catalog>
```

For example, a reference to the schema URI "httpd://pds.nasa.gov/pds4/pds/v1/PDS4_PDS_1301.xsd" will be translated to the local file reference "/n/sbnops/lcltools/schema/pds/v1/PDS4_PDS_1301.xsd".

uri

This catalog file adds `uri` statements **before** the `rewriteURI` statement to catch references to mission (EPOXI) schema files still in local development. The `uri` statements have to come first because the catalog processor will stop with the first statement that matches - so in this case if the `rewriteURI` statement came first, the processor would never make it past there.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE catalog PUBLIC "-//OASIS//DTD XML Catalogs V1.1//EN" "http://www.oasis-
open.org/committees/entity/release/1.1/catalog.dtd">
<catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog">
  <uri name="http://pds.nasa.gov/pds4/mission/epoxi/v1/EPOXI_1-0.xsd"
       uri="file:///home/raugh/Oxygen/epoxiDD/epoxi_draft.xsd"/>
  <uri name="http://pds.nasa.gov/pds4/mission/epoxi/v1/EPOXI_1-0.sch"
       uri="file:///home/raugh/Oxygen/epoxiDD/epoxi_draft.sch"/>
  <rewriteURI uriStartString="http://pds.nasa.gov/pds4"
              rewritePrefix="file:///n/sbnops/lcltools/schema"/>
</catalog>
```

The `uri` statement replaces the entire matched `name` with the associated URI value, so the string "http://pds.nasa.gov/pds4/mission/epoxi/v1/EPOXI_1-0.sch", for example, will be replaced by the reference to the local file "/home/raugh/Oxygen/epoxiDD/epoxi_draft.sch".

delegateURI

Alternately, if you are working with several mission dictionaries in active development scattered across your disc space or network, you might want to use a catalog file specifically to handle the

mission dictionaries and use your local schema tree for the rest. In that case, you would use a delegateURI statement to trap references to all mission namespaces and pass them off to a different catalog file, while the rest fall through to be handled by the rewriteURI:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE catalog PUBLIC "-//OASIS//DTD XML Catalogs V1.1//EN" "http://www.oasis-
open.org/committees/entity/release/1.1/catalog.dtd">
<catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog">
  <delegateURI uriStartString="http://pds.nasa.gov/pds4/mission"
          catalog="file:///home/raugh/Oxygen/XMLCatalogs/mission_schemas.xml"/>
  <rewriteURI uriStartString="http://pds.nasa.gov/pds4"
          rewritePrefix="file:///n/sbnops/lcltools/schema"/>
</catalog>
```

In this case, all references beginning with "http://pds.nasa.gov/pds4/mission" will be passed to the "mission_schemas.xml" catalog file for resolution. Say that catalog file looks like this:

*Contents of **mission_schemas.xml**:*

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE catalog PUBLIC "-//OASIS//DTD XML Catalogs V1.1//EN" "http://www.oasis-
open.org/committees/entity/release/1.1/catalog.dtd">
<catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog">
  <rewriteURI uriStartString="http://pds.nasa.gov/pds4/mission/epoxi/v1"
          rewritePrefix="file:///home/raugh/Oxygen/epoxiDD"/>
  <rewriteURI uriStartString="http://pds.nasa.gov/pds4"
          rewritePrefix="file:///n/sbnops/lcltools/schema"/>
</catalog>
```

Given these two catalog files (in their proper places, of course):

- a reference to "http://pds.nasa.gov/pds4/mission/epoxi/v1/EPOXI_1-0.xsd" will be passed to *mission_schemas.xml*, which will return a new value of "file:///home/raugh/Oxygen/epoxiDD/EXPOXI_1-0.xsd";
- a reference to "http://pds.nasa.gov/pds4/mission/di/v1/DEEP_IMPACT_1-1.xsd" will also be passed to *mission_schemas.xml*, but will return a new value of "file:///n/sbnops/lcltools/schema/mission/di/v1/DEEP_IMPACT_1-1.xsd"; and
- a reference to "http://pds.nasa.gov/pds4/pds/v1/PDS4_PDS_1300.xsd" will be handled by the first catalog file, and will return a value of "file:///n/sbnops/lcltools/schema/pds/v1/PDS4_PDS_1300.xsd".

References

Here are some links to the various standards mentioned above:

- XML Catalogs V1.0, October 2002

- XML Catalogs V1.1, October 2005
- OASIS Technical Resolution 9401:1997 (pre-XML catalogs)
- Extensible Markup Language (XML) 1.0 (Fifth Edition)
- XML Schema Definition Language (XSD) 1.1 Part 1: Structures