

Using the LEAD Schema And Storing Metadata in MyLEAD

Scott Jensen
Indiana University
3/15/2006

Abstract	1
LEAD Schema	1
Metadata Attributes and Elements	1
An example:.....	2
What Is Required	2
Minimal LEAD metadata document:.....	2
Optional Metadata.....	4
Dynamic Metadata Attributes	4
Two examples:.....	5
Queryable Metadata	9
Undefined dynamic attributes:.....	9
Adding Metadata to Existing Documents	9
What can be passed in the add method:.....	10
Always valid:.....	10
Valid if not already in the document:.....	10
Valid with restrictions:.....	10
Updating Metadata in myLEAD.....	11

Abstract

The focus of this paper is two-fold; first we discuss the required and optional elements of the LEAD schema as well as the minimal requirements for a metadata document to validate against the LEAD schema. The second focus of the paper is on storing metadata based on the LEAD schema in myLEAD with an emphasis on storing model parameters as dynamic metadata attributes.

LEAD Schema

The following discussion of storing metadata in myLEAD often refers to the color-coded, 4-page printout of the LEAD schema that is attached with this document. Each metadata XML document that complies with the LEAD schema starts with a LEADresource XML element as shown on page 1 of the printout. Aside from the file or collection's global ID, all of the metadata is contained within the XML element named "data" which starts on page 2 of the printout.

Metadata Attributes and Elements

From a myLEAD perspective, metadata is composed of metadata attributes and elements. These are not directly XML attributes and elements. Sorry for the overloaded terms – we started this terminology based on MCS attributes. Every queryable metadata value,

whether a string, number, or date is stored in a metadata element, and metadata attributes are used to group these elements into concepts. In the discussion below, I use either “XML element” or “node” when referring to an XML element in the schema, and the terms “metadata attribute” and “metadata element” when referring to the attributes and elements stored in myLEAD.

In myLEAD we look at metadata attributes and elements as being either structural or dynamic. The structural attributes are those defined by the structure of the LEAD schema. In the color-coded schema printout, the schema elements that are in **blue** are structural metadata attributes and the XML elements under them in **red** are structural metadata elements. For now, if you are skimming over the schema, skip over the XML element named “detailed”; it will be covered later in the discussion of dynamic metadata attributes. You may have also noticed the patriotic red and blue XML elements such as **resourceID**, **acceconst**, and **useconst**. Each metadata element is contained within a metadata attribute in myLEAD, but these XML elements represent a concept in the schema that only has one XML element, so it is both a metadata attribute and metadata element when stored in myLEAD.

An example:

On page 2 of the schema printout, there is a blue XML element named **descript** that is a structural metadata attribute and it contains three red XML elements (**abstract**, **purpose**, and **supplinf**), which are metadata elements. Based on the FGDC workbook, each of these elements is “free text”, but the workbook provides some guidelines as to what each of these should contain. Also, since the border of the supplinf XML element is dashed, it is an optional metadata element – a LEAD metadata document can contain a descript metadata attribute that contains only two metadata elements – abstract and purpose, but it must contain these two.

What Is Required

Must a LEAD metadata document contain a descript XML element with at least the abstract and purpose child XML elements? Yes it must. If we flip back to page 1 of the printout, you will see that the “root” of the document, LEADresource, has two children – the resourceID which contains the global ID and a “choice” symbol (the thing that looks like a tipped over bottle with an electric switch inside). Since this choice is not optional, a LEAD document must contain either a “data” child or a “workflow” child. Currently we are only using the data path in the LEAD schema. Since the data XML element is required, we need to include it and any required children. The data node has two required child nodes: idinfo and metainfo, as well as four subsequent optional children (geospatial, distinfo, dataqual, and enclosedresources). Since the idinfo node is required, its six children (none of which are optional) must also be in each document – one of these children is the descript node.

Minimal LEAD metadata document:

As discussed above, there are some metadata that each LEAD metadata document must have in order to validate against the LEAD schema (such as abstract and purpose within descript). The minimal metadata document must contain the global ID (resourceID) and

the required nodes under idinfo and metainfo within the data node. Following is a list of the minimal metadata elements arranged in schema order and grouped according to their metadata attribute. Also noted in this list is whether the document should contain exactly one instance of the metadata element (1) or whether multiple instances are allowed.

resourceID – the global ID (1)

citation:

origin (1 or more)

pubdate (1)

title (1)

descript:

abstract (1)

purpose (1)

status:

progress (1)

update (1)

acconst – access constraints (1)

useconst – usage constraints (1)

theme: (there can be 1 or more instances of theme)

themekt – this is a reference to the thesaurus defining the keywords in the theme (1)

themekey – common theme or word describing the data (1 or more)

metainfo:

metd – date the metadata were created or updated (1)

metstdn – Name of the metadata standard used (1)

metstdv – version of the standard named in metstdn (1)

In all, there are only 14 metadata elements that must be included in each LEAD metadata document aside from the global ID. Each of these metadata attributes except metainfo and citation is described in section one of the FGDC schema; metainfo is discussed in section ten and citation is covered in section eight. Of these metadata elements, some will be standard, such as the metstdn – for which the FGDC documentation suggests using “FGDC Content Standard for Digital Geospatial Metadata”. We need to determine whether we are using the FGDC standard or whether we have our own LEAD standard (and what it’s name and version number are). If there are no access or use constraints, then the FGDC standard suggests “none” for acconst and useconst. Likewise, possible values for pubdate in the FGDC documentation are “Unknown” or “Unpublished material”, status can be “In work” “Planned”, or “Complete” and the progress can be “Unknown”.

The keywords can come from a number of sources, but to the extent that established vocabularies are used, queries will be more accurate and the ontology service can add greater value to the query process.

Optional Metadata

Within the children of the idinfo node there are some optional nodes in the schema. The most likely to be used are the place, stratum, and temporal keyword metadata attributes. For example, in a prior LEAD AG meeting, Rich was describing an experiment looking at the Great Plains, which could be included as a “place” keyword. However, it’s important to note that optional nodes usually contain at least some required child nodes – an example is the place keyword which must contain both a thesaurus reference and one or more keywords.

Most of the optional metadata in the LEAD schema is in three of the four optional child nodes under data: geospatial, distinfo, and dataqual. The dataqual node contains information regarding the lineage of the file or collection - see Yogesh regarding the provenance data included in that section of the schema. The distinfo node contains information regarding the format in which the data is distributed – including file format, format version, file size, and compression. However, keep in mind that within distinfo all of that information is under the optional stdorder node, but the distinfo node also has a required cntinfo metadata attribute (contact info) child node. This can contain extensive contact information, but the only required metadata elements are your name, phone number, city, state, and zip code.

The last remaining optional node under data is geospatial. This node contains the spatial and temporal information as well as the dynamic metadata attributes. If a metadata document includes temporal or spatial data, then it must also contain the other – the timeperd and bounding under spdom (spatial domain) are both required nodes in the schema (if you include the optional geospatial node). The bounding box uses east, west, north, and south lat/lon coordinates, so if metadata is being generated for a workflow or model that uses a center lat/lon, the metadata generator will need to determine the bounding box if spatial data is to be included in the metadata.

Dynamic Metadata Attributes

Not all of the possible metadata could be captured in the LEAD schema, and even if it could be, the schema would become unwieldy. In myLEAD, all of the metadata is stored based on metadata attributes and metadata elements. For the metadata defined by the LEAD schema structure, the metadata attributes and elements were defined based on the tags of the nodes in the schema. If we want to capture model parameters, these need to be stored as dynamic metadata attributes. Each such attribute starts as a detailed node (under the eainfo node). The structure under the eainfo node may look a bit convoluted, but basically it just says that you can have detailed nodes, overview nodes, or a combination of both – just keep in mind all of the detail nodes must be before the overview nodes in a LEAD metadata document.

Each instance of the detailed node is an instance of a dynamic metadata attribute. Since we need to allow for complex metadata attributes, keep in mind that a metadata attribute may have sub attributes. Under the detailed node, the **entyp** and **entypds** nodes (under entyp) provide the name and source of the dynamic metadata attribute. Whereas for

structural attributes the name is based on the tag of the node in the schema, for dynamic metadata attributes enttypl contains the name. The enttypls is the source of the definition for the name. This is needed because the same name could be used for slightly different metadata attributes in different models – such as the WRF model compared to data mining. As with structural metadata attributes, each dynamic metadata attribute can contain elements or subattributes – both are represented by attr child nodes of the detail node. The **attrlabl** and **attrdefs** nodes under each attr node contain the name and source of the metadata element or subattribute – similar to enttypl and enttypls. One of the children of each attr node is a “choice”. If the attr node represents a metadata element, then the attrv child nodes contain the values. If the attr node represents a subattribute, then its “choice” would be attr child nodes that would represent the elements within the subattribute and/or another level of subattributes.

Two examples:

Suresh provided a copy of an ARPS namelist input file. If scientists wanted to capture some of these parameters as metadata of an experiment, then dynamic metadata attributes could be defined for some (or all) of the parameters in the namelist file. Additional dynamic metadata attributes can be defined over time. When dynamic metadata attributes are defined, we also identify their data types – making more powerful and efficient queries possible.

In the ARPS file we will look at two namelist parameters: grid, and the parameters that make up the ADAS analysis type and correlation specification. The grid parameter was selected because Suresh had identified it as one of the most relevant parameters. The other set of parameters will be used to illustrate using subattributes.

From the ARPS file:

```
&grid
  dx          = 1000.000,
  dy          = 1000.000,
  dz          = 500.000,
  strhopt     = 0,
  dzmin       = 100.000,
  zrefsfc     = 0.0,
  dlayer1     = 0.0,
  dlayer2     = 1.0e5,
  strhtune    = 1.0,
  zflat       = 1.0e5,
  ctrlat      = 36.00,
  ctrlon      = -100.00,
/
```

Based on the namelist file provided, the “source” for all metadata attributes and elements (**enttypls** and **attrdefs** respectively) will be “ARPS 5.2.0”.

Creating a dynamic metadata attribute based on the above snippet from the namelist file, we will have a metadata attribute named “grid” and a subattribute named “gridStretch”. Within the grid metadata attribute there will be the following metadata elements: dx, dy,

dz, zflat, ctrlat, and ctrlon (all using the float data type). Within the gridStretch subattribute, there will be metadata elements for strhopt, dzmin, zrefsfc, dlayer1, dlayer2, and strhtune (again, all using the float data type - except strhopt which would be an integer). Within a LEAD metadata document, a detail node such as the following could be included based on this definition (omitting some elements):

```

<detailed>
  <enttyp>
    <enttyp1>grid</enttyp1><enttypds>ARPS 5.2.0</enttypds>
  </enttyp>
  <attr>
    <attrlabl>gridStretch</attrlabl><attrdefs>ARPS 5.2.0</attrdefs>
    <attr>
      <attrlabl>strhopt</attrlabl><attrdefs>ARPS 5.2.0</attrdefs>
      <attrv>2</attrv>
    </attr>
    <attr>
      <attrlabl>dzmin</attrlabl><attrdefs>ARPS 5.2.0</attrdefs>
      <attrv>100.000</attrv>
    </attr>
  </attr>
  <attr>
    <attrlabl>dx</attrlabl><attrdefs>ARPS 5.2.0</attrdefs>
    <attrv>1000.000</attrv>
  </attr>
  <attr>
    <attrlabl>dz</attrlabl><attrdefs>ARPS 5.2.0</attrdefs>
    <attrv>500.000</attrv>
  </attr>
</detailed>

```

This illustrates storing a namelist parameter as a metadata attribute. Since attr nodes within a “detailed” node in the schema are optional, a dynamic metadata attribute could be defined for a namelist parameter with subattributes and metadata elements for all of the variables in that parameter, but each use of that metadata attribute could omit some of the subattributes and metadata elements – including only those relevant to the particular experiment.

The other example we will look at is the ADAS analysis parameters. These are actually a set of namelist parameters that are included as arrays of values with each position in the array representing a pass. In the file we are using there are values for 8 passes. Are there always eight passes or can this vary? For purposes of this example, we will assume this could vary. There is also a distance parameter that applies to all passes. For this example, we will create a dynamic metadata attribute named “AdasAnalysis” which will contain a subattribute named “AdasPass” for each pass, and an element named “sfcqcrng” which is the name of the namelist parameter for the horizontal distance parameter applicable to all passes. Within the subattribute there will be metadata elements defined for the pass number (the array index in the namelist file), ianxtyp (analysis type), xyrange (horizontal range), zrange, and thrange. An example “detail” node for this dynamic metadata attribute could be as follows (with only 2 passes and some elements omitted):

```

<detailed>
  <enttyp>
    <enttyp1>AdasAnalysis</enttyp1><enttypds>ARPS 5.2.0</enttypds>
  </enttyp>
  <attr>
    <attrlabl>sfcqcrng</attrlabl><attrdefs>ARPS 5.2.0</attrdefs>
    <attrv>100000.000</attrv>
  </attr>
  <attr>
    <attrlabl>AdasPass</attrlabl><attrdefs>ARPS 5.2.0</attrdefs>
    <attr>
      <attrlabl>pass</attrlabl><attrdefs>ARPS 5.2.0</attrdefs>
      <attrv>1</attrv>
    </attr>
    <attr>
      <attrlabl>ianxtyp</attrlabl><attrdefs>ARPS 5.2.0</attrdefs>
      <attrv>11</attrv>
    </attr>
    <attr>
      <attrlabl>xyrange</attrlabl><attrdefs>ARPS 5.2.0</attrdefs>
      <attrv>300000</attrv>
    </attr>
    <attr>
      <attrlabl>zrange</attrlabl><attrdefs>ARPS 5.2.0</attrdefs>
      <attrv>500</attrv>
    </attr>
  </attr>
  <attr>
    <attrlabl>AdasPass</attrlabl><attrdefs>ARPS 5.2.0</attrdefs>
    <attr>
      <attrlabl>pass</attrlabl><attrdefs>ARPS 5.2.0</attrdefs>
      <attrv>2</attrv>
    </attr>
    <attr>
      <attrlabl>ianxtyp</attrlabl><attrdefs>ARPS 5.2.0</attrdefs>
      <attrv>21</attrv>
    </attr>
    <attr>
      <attrlabl>xyrange</attrlabl><attrdefs>ARPS 5.2.0</attrdefs>
      <attrv>120000</attrv>
    </attr>
    <attr>
      <attrlabl>zrange</attrlabl><attrdefs>ARPS 5.2.0</attrdefs>
      <attrv>300</attrv>
    </attr>
  </attr>
</detailed>

```

There is also a `kpvar` array that has parameters that seem to apply to all of the passes. These could be defined as metadata elements within the `AdasAnalysis` metadata attribute. While these examples show the flexibility we have in defining metadata attributes and elements, they also illustrate why the meteorological researchers and researchers building metadata generators need to be involved in defining dynamic metadata attributes. The structural attributes have been defined in myLEAD based on the structure of the schema, but to define the relevant dynamic metadata attributes, we need input as to not only which

parameters should be included in the metadata, but also what names and sources would be meaningful to users as well as what structure works well for metadata generators.

Queryable Metadata

All of the metadata contained in a LEAD document is saved in myLEAD and included in query responses. However, not every leaf node in the LEAD schema is defined as a metadata element. An example is the optional `dsgpoly` node under the `spdom` node. In addition to the required bounding box coordinates for the spatial domain, a metadata document could contain more precise polygon definitions within the bounding box. In the FGDC documentation, these are referred to as G-rings and they use the state of Virginia as an example. The bounding coordinates contained in the bounding metadata attribute would provide the bounding box around the state, while G-rings could be defined for the boundary of the state and associated islands (Virginia has a jagged, non-rectangular shape). The `dsgpoly` nodes can actually either be an “outer” G-ring (such as the border of a state) or a “hole” within an outer G-ring. If the LEAD community finds at a later date that it is generating these optional `dsgpoly` nodes, new metadata attributes could be defined and the existing data could be processed since every blue node in the schema printout is saved as a CLOB.

Although the `dsgpoly` nodes are not saved as queryable metadata elements, an additional spatial metadata attribute is defined in myLEAD based on the bounding box coordinates. This metadata attribute is created when an `spdom` node is included in the metadata document. Users can then issue spatial queries against this metadata element (such as whether it contains a specific lat/lon point).

Undefined dynamic attributes:

Another example of a situation in which the metadata is stored but not queryable is when a detailed node contains `enttypl`, `enttylds`, `attrlabl`, or `attrdefs` values that do not match a defined dynamic attribute. However, myLEAD will include detailed node data to the extent that it is defined. This could be useful in cases where a researcher building a model wants to include some additional metadata within a defined dynamic attribute – possibly for potential extension of the model. The full detail node would be stored and the defined dynamic metadata elements would be queryable.

When storing metadata attributes and elements, as long as all of the CLOBs are successfully saved to the database then no error is generated if some of the metadata could not be stored in the defined metadata attributes and elements.

Adding Metadata to Existing Documents

So far the discussion has focused on adding new metadata documents to myLEAD, but metadata can also be added to an existing document. Metadata attributes can be added for any metadata attribute that allows multiple instances or any optional node (if it is not already in the document). Instead of taking a document rooted at the `LEADresource` node, the `add` method will take a LEAD schema fragment rooted at any node for which a metadata attribute can be added. In addition, the fragment passed can also be a fragment rooted at any higher optional node for which no metadata attribute children exist in the document.

Based on these properties, the LEAD XML fragment passed to the add method could be any of the keyword type nodes (theme, place, stratum, or temporal) since each of these is a metadata attribute that allows multiple instances. If the document already has temporal (timeperd) and spatial (bounding) metadata attributes, then the optional vertdom (vertical spatial domain) could be passed as the XML fragment – assuming the document does not already have a vertdom node. Note that a vertdom node cannot be added if the document does not already contain both timeperd and bounding metadata attributes. Possibly less obvious, the detailed nodes used for dynamic metadata attributes cannot be added unless the document already has spatial and temporal metadata (since the nodes for those metadata attributes are on a required path within the geospatial node). A higher node, such as geospatial, can also be passed as the XML fragment if the document does not already contain any of its children. In contrast, the metainfo node or descriptor nodes cannot be added since these are required nodes in the LEAD schema and must already exist in the document.

What can be passed in the add method:

This list is divided into the following categories: always valid, valid if does not already exist, and those restricted based on other metadata attributes. Although probably obvious, the XML fragment must always validate against the LEAD schema for the subtree rooted at that node – for example, a geospatial node must always contain temporal and spatial metadata.

Always valid:

theme
place
stratum
temporal
resourceID (within the enclosedresources node)

Valid if not already in the document:

geospatial
idinfo this is the idinfo under geospatial. Since it is the only required child node of geospatial, it is the exception to having to be optional.
distinfo
cntinfo as with idinfo above, since this is the only node on the only required child path within distinfo, it can also be passed to the add method. The distrib node is not included in the list because it is redundant.
dataqual
enclosedresources

Valid with restrictions:

dsgpoly only if the bounding metadata attribute exists.
vertdom only if the timeperd metadata attribute exists.
eainfo only if the timeperd metadata attribute exists and no detailed or overview metadata attributes exist.
detailed only if the timeperd metadata attribute exists.

overview only if the timeperd metadata attribute exists.
stdorder only if the cntinfo metadata attribute exists.
procstep only if a procstep node already exists. Since a dataqual node requires both a
 “complete” child node and a procstep node, procsteps can only be added if
 they are not the first one.

When metadata attributes are added that allow for multiple instances, the instance being added will always be appended. For example, if a LEAD metadata document had two theme nodes when originally added to myLEAD, and another theme was added later, the theme added would be the third theme node when the document is returned in response to a query.

Updating Metadata in myLEAD

The update method in myLEAD is actually a replace/add at the metadata attribute level, so any node in the schema that is at the metadata attribute level or higher (an ancestor of a metadata attribute node) can be passed in the update method. The only requirement is that the node already exists (e.g., a geospatial node exists if the document has a timeperd metadata attribute). The one exception to this is the resourceID node that is a direct child of the root LEADresource node of the document. That resourceID node contains the document’s global ID and changing it could effectively orphan files stored in RLS/DRS.