MyLead Release V1.3.8
myLEAD Developer’s Guide
1 Introduction................................................................................................................. 2
  1.1 myLEAD License ............................................................................................... 2
2 Exceptions, Return Types and Conventions ............................................................... 3
3 Creating the myLEAD Client (LeadClient).......................................................... 3
4 Creating an Instance of the LeadClient Class....................................................... 4
5 Setting Up a myLEAD User ....................................................................................... 5
  5.1 Setting Up a myLEAD User ............................................................................... 5
  5.2 Checking if a User Exists.................................................................................... 6
6 Adding Metadata to myLEAD using the myLEAD Client......................................... 6
  6.1 Adding Metadata Attributes to Existing Objects in myLEAD ........................... 8
  6.2 Updating Metadata in Existing Objects ............................................................ 10
7 Deleting Metadata Objects........................................................................................ 13
8 Deleting Metadata Attributes Within Objects........................................................... 13
9 Moving Metadata Objects in the Hierarchy.............................................................. 15
  9.1 Determining a User’s Whiteboard Project........................................................ 16
10 Querying the myLEAD Metadata Catalog ........................................................... 17
  10.1 Query Response Format.................................................................................... 17
  10.2 Context Queries ................................................................................................ 18
    10.2.1 Defining Metadata Attribute and Element Criteria For a Query .............. 21
  10.3 Query By Object ID.......................................................................................... 24
  10.4 Query Filters ..................................................................................................... 26
    10.4.1 Hierarchy Filters ....................................................................................... 26
    10.4.2 Content Filters........................................................................................... 28
  10.5 Query Exclusions.............................................................................................. 32
Appendix A: Metadata Attribute and Element Names .................................................... 35
Appendix B – LEAD Metadata Schema (LMS) .............................................................. 39
1 Introduction
This document provides an overview of using the myLEAD Client Toolkit to add, delete, update, or query metadata in the myLEAD Server metadata catalog. This document assumes that you have already installed the myLEAD client and server with all of the software they are dependent on as discussed in the installation guide.

1.1 myLEAD License
The file doc/myLEAD-Licence.txt within the source distribution directory contains the product license. Make sure that you read this license and accept its conditions before continuing.
2 Exceptions, Return Types and Conventions

All of the public methods of the myLEAD client API can throw an Exception of type MyLeadException, and all of the methods return a status indicator of the type ReturnType.

Many of the methods for adding metadata or updating metadata take String parameters that include either LEAD Metadata Schema (LMS) compliant documents or fragments from such documents. In the examples, the full documents are often shown as: `<LEADresource/>` and fragments that are metadata attributes (explained later) are shown as `<MetadataAttribute/>`.

Files, Aggregations, and Objects:
Metadata is stored in myLEAD regarding files and aggregates – collections, experiments, and projects. In this documentation, if a method can take any of these types, it will be referred to as a myLEAD object.

Metadata Attributes, Metadata Elements, Nodes, and XML Attributes:
In this document we refer to metadata attributes and metadata elements. These are not attributes and elements in the XML sense. Metadata attributes are a subtree in the LMS that define a concept – such as the distinfo subtree which contains all information regarding the distribution of a particular data product, including the format of an object and contact information regarding the organization that created the object. To avoid confusion with metadata attributes or the metadata elements they contain, elements in the LMS will generally be referred to as nodes in this documentation. Likewise, whenever the documentation is referring to an attribute in an XML element, it will be referred to as an XML attribute to avoid confusing it with a metadata attribute in myLEAD.

Although the methods in myLEAD use String parameters for LMS documents or fragments of the LMS based on metadata attributes, both entire LMS compliant documents as well as fragments for nodes within the LMS can be created using the document classes within XML Beans. The toString() method of the bean can then be called when passing it as a parameter to a myLEAD method.

An XML Bean version of the LMS can be downloaded from:
http://www.extreme.indiana.edu/rescat/metadata/

In addition, there are only 14 required nodes in the LMS, and a utility that creates a LMS XML bean with default settings can be downloaded from the same page – see the section titled “LEAD Metadata Schema Utility”.

3 Creating the myLEAD Client (LeadClient)

The myLEAD client API class is named LeadClient and will generally be accessed through the myLEAD Agent, but this guide provides information on using the myLEAD client directly using the LeadClient class. The API handles setting up the Grid Data
Service (GDS) using OGSA-DAI and creating the necessary OGSA-DAI perform documents. This documentation covers the Java API for the LeadClient class and does not cover the underlying details of the perform document activity schemas.

4 Creating an Instance of the LeadClient Class

When using the LeadClient, the following packages need to be included in your Java code:

```java
edu.indiana.dde.mylead.client.*
edu.indiana.dde.mylead.common.*
edu.indiana.dde.mylead.response.metadata.*
```

The first two of these packages are included in the leaddai-X.X.X.jar file that is installed as part of the myLEAD Client installation (where X.X.X. is the version number for the LeadClient). The third package is in the MyLeadResponse-X.X.X.jar (where X.X.X is the version of the response jar).

In your program code, you can create an instance of LeadClient with either of the following two constructor methods:

```java
LeadClient (String gridfactoryurl);
LeadClient (String gridfactoryurl, int servicetime);
```

The only parameter required in the first version of the constructor is the factory URL. Following is an example of the factory URL – you will need to modify it for your server name and port (as defined when OGSA-DAI and the myLEAD server are installed).

```text
http://localhost:8080/ogsa/services/ogsadai/MyLeadGDSF
```

If you want to have a lifespan for the client that is different from the default lifespan of 5 minutes, the second version of the constructor can be used and the number of minutes for the life span is specified using minutes as the servicetime parameter.

After an instance of LeadClient is created, it can be used to define, update, or delete users, add, update, and delete metadata for files and collections, query metadata, and perform some administrative functions. The interface for the myLEAD Client is defined in the LeadClientIntf class in the edu.indiana.dde.mylead.client package.

Although the LeadClient is assigned a lifetime when created, if it is no longer needed, its close method should be called to conserve resources.
Example: Creating a myLEAD Client

```java
import edu.indiana.dde.mylead.client.*;
...
String host = "localhost";
String port = "8080";
...
String factory = "http://" + host + ":" + port + "/ogsa/services/ogsadai/MyLeadGDSF";
LeadClient client = new LeadClient(factory);
...
client.close();
```

5 Setting Up a myLEAD User

Before you can enter any data in the myLEAD catalog using the myLEAD Client, you will need to have defined at least one user in the myLEAD database. If a user is added through the LEAD portal, then a myLEAD user will automatically be created and there is no need to separately call this method.

5.1 Setting Up a myLEAD User

If a user definition needs to be added without using the LEAD portal, a new user can be added by calling the following method in `LeadClient`:

```java
create(String dn, MyLeadUser user);
```

**Parameters:**

dn:
The dn parameter is the distinguished name of the user who is adding the user (this user must already be defined in myLEAD).

user:
This parameter is an instance of the `MyLeadUser` class. The `MyLeadUser` class is declared in the package `edu.indiana.dde.myLEAD.common` in the `leaddai` jar file. This class has variables for all of the possible user settings, but the default constructor sets most of these fields to empty strings. Only the dn and the user’s name actually need to be set in the `MyLeadUser` class using the following methods:

```java
void setDn (String dnStr);
void setName (String nameStr);
```

For all other possible user settings please refer to the JavaDocs for the `MyLeadUser` class.
Example: Creating A User

```java
import edu.indiana.dde.mylead.client.*;
import edu.indiana.dde.mylead.common.*;

... <LeadClient created as client> ...

MyLeadUser user = new MyLeadUser();
user.setDn("/C=US/O=Town of Bedrock/CN=Fred Flinstone");
user.setName("Fred Flinstone");

... Setting other user properties ...

String adminDn = "/C=US/O=Lead/CN=Lead Administrator";
ReturnType ret = client.create(adminDn, user);
```

When a user is created, myLEAD automatically creates a “whiteboard” project in the user’s workspace with the name “Default Project”. When the user imports data from the public data catalog it is automatically stored in the whiteboard project. In addition, a collection named “Workflow Templates” is automatically created within the whiteboard project.

5.2 Checking if a User Exists

Each user is identified in myLEAD by their DN. Whether a user has been defined in myLEAD can be determined by calling the userExists method with their DN as follows:

```java
userExists (String dn)
```

This method call will return a boolean indicating if the user exists in myLEAD (true) or not (false).

6 Adding Metadata to myLEAD using the myLEAD Client

Files, collections, experiments, and projects (also referred to in this documentation as objects) are added to the metadata catalog using the create method. Objects can only be added for users defined in myLEAD either through the portal or through myLEAD. The create methods for adding objects are:

```java
create(String dn, String item, String objectType)
create(String dn, String item, String objectType, String parentGuid)
create(String dn, CreateDataHolder[] myLeadHolder)
```

Parameters:

dn:
In all three versions, the dn parameter is the distinguished name of the user who is creating the file or collection being added.
Item:
This is a String variable that must contain an XML document that validates against
the LEAD Metadata Schema (LMS). Any call to create which does not contain an item
that validates against the LMS will be rejected. Although namespaces must be used in
the document, as long as the document is valid, any namespace prefixes can be used.

objectType:
This parameter must match one of the four types currently defined in myLEAD:
- FILE
- COLLECTION
- EXPERIMENT
- PROJECT
If the object being created is a PROJECT, then no parent should be specified. If a parent
is specified for a project, the return type will be NO_PARENT_ALLOWED. For all
object types other than projects, a parent must be specified. If no parent is specified
when one is required, the return type is NO_PARENT_SPECIFIED. The parent’s type
must be a higher level in the hierarchy, with the exception that collections can contain
other collections. If the parent is not valid (e.g., a collection is specified as the parent of
an experiment being added) then the return type is INVALID_PARENT.

parentGuid:
This is the global ID of the parent object. For example, all files must belong to a
collection, experiment, or project, so if a file is being added, the parent’s global ID is
specified for this parameter. The global ID is the resourceId element of the LMS,
which is a direct child of the LEADresource element that is the root of every LMS
document.

myLeadHolder:
This is an array of CreateDataHolder instances. This class is defined in the same
package as the LeadClient class and is included in the leaddai jar file. Each instance
of the CreateDataHolder class contains String variables for the item,
objectType, and parentGuid. The version of the create method which includes
this parameter allows the user to add multiple objects for a single user in one method
call. In addition to the default constructor, it has the following constructor:

CreateDataHolder (String myItem, String myType, String pGuid)

When objects are added to myLEAD, the system automatically records the time and date
when they were added.
Example: Creating a Project Object

```java
import edu.indiana.dde.mylead.client.*;
import edu.indiana.dde.mylead.common.*;
...
String dn = "/C=US/O=Town of Bedrock/CN=Fred Flinstone";
...
... <LeadClient created as client> ...
...
ReturnType ret = client.create (dn, "<LEADresource/>", "PROJECT");
Client.close();
```

The above example creates a project, which is the highest level in the LEAD hierarchy, so no parent object needs to be specified.

6.1 Adding Metadata Attributes to Existing Objects in myLEAD

After an object has been added to myLEAD, additional metadata attributes can be added for that object if allowed in the LMS. The `addAttribute` method in `LeadClient` is used to add metadata to existing objects:

```java
addAttribute(String dn, String objectGuid, String item)
addAttribute(String dn, String objectGuid, String items[])
addAttribute(String dn, CreateAttrHolder[] items)
```

**Parameters:**

dn:
As in the `create` methods, the `dn` parameter in each of the `addAttribute` methods is the distinguished name of the user who owns the object which metadata is being added to.

objectGuid:
This is the global ID (`resourceID` in the LMS) for the object the metadata should be added to. The object identified by this ID must already exist in myLEAD and the user identified by the `dn` parameter must own this object.

item/items (String):
In the first two versions of the `addAttribute` method the `item` parameter is a `String`, and that string must be a valid XML fragment for a subtree in the LMS for one of the following nodes:

Within LEADresource/data/idinfo:
- keywords/theme
- keywords/place
- keywords/stratum
- keywords/temporal

LEADresource/data/geospatial (only if geospatial does not already exist)
Within LEADresource/data/geospatial:
- idinfo  (only if geospatial does not already exist)
- idinfo/spdom/dsgpoly
- idinfo/vertdom  (only if vertdom does not already exist)

LEADresource/data/eainfo  (only if eainfo does not already exist)

Within LEADresource/data/eainfo:
- detailed
- overview

LEADresource/data/distinfo  (only if distinfo does not exist already)

Within LEADresource/data/distinfo:
- cntinfo (only if distinfo does not exist already)
- stdorder

LEADresource/data/dataqual  (only if dataqual does not exist already)

Within LEADresource/data/dataqual:
- lineage/procstep  (only IF dataqual already exists)

Some of these nodes are optional, but can only be added if they did not exist already in the LMS for an object. These include geospatial, vertdom, eainfo, distinfo, cntinfo, and dataqual. As discussed below, there are separate methods that can be used to update existing metadata for some of these nodes.

The fragments passed as item parameters must be namespace qualified, but the namespace prefixes used in the fragments can differ from the prefixes used in the original document (and could also vary between fragments when using version of addAttribute which takes an array of CreateAttrHolder instances).

Certain nodes in the LEAD schema have been identified as concepts, and these are referred to as “metadata attributes” in myLEAD. In the schema diagram of Appendix B the nodes with blue (non-italicized) tags are metadata attributes. When adding metadata to an existing object or updating metadata, the metadata attribute node is the smallest unit of metadata that can be added or updated since these define a single concept in the LMS.

Items (CreateAttrHolder):
The version of the addAttribute method which takes an array of CreateAttrHolder instances provides the ability to add multiple metadata fragments in one call – even to different metadata objects as long as they are all owned by the same user. The CreateAttrHolder class is defined in the same package as LeadClient and is in the leaddai jar. This holder class just has variables for the item
and objectGuid. In addition to the default constructor, it has the following constructor available:
CreateAttrHolder (String myItem, String guid)

**Example: Adding Metadata Attributes**

```java
import edu.indiana.dde.mylead.client.*;
import edu.indiana.dde.mylead.common.*;

... String dn = "/C=US/O=Town of Bedrock/CN=Fred Flinstone";
String guid = "urn:uuid:678580d4-6a48-1030-842e-000bdba158a1";
... <LeadClient created as client> ...
... ReturnType ret = client.addAttribute (dn, guid, "<MetadataAttribute/>");
Client.close();
```

In the above example, the global ID of the object to which the metadata attribute is being added is set to a literal value, but normally it would be passed into the method as a parameter.

### 6.2 Updating Metadata in Existing Objects

Existing metadata for an object in myLEAD can be updated using the following LeadClient methods:

- `updateAttribute(String dn, String objectGuid, String item, Integer position, Long lastUpdate)`
- `updateAttribute(String dn, UpdateAttrHolder[] updates)`

**Parameters:**

dn:
As in the create and addAttribute methods, the dn parameter in each of the updateAttribute methods is the distinguished name of the user who owns the object for which metadata is being updated.

objectGuid:
This is the global ID (resourceID in the LMS) of the object for which the metadata is being updated. The object identified by this ID must already exist in myLEAD and the user identified by the dn parameter must own this object.

Item:
This is a String containing an XML fragment which will validate against the LMS and which is rooted at one of the nodes in the LMS which has been identified as a metadata attribute. These are the non-italicized blue nodes in the diagram of Appendix B. Following is a list of these metadata attribute nodes:

Within LEADresource/data/idinfo:
- citation
- descript
- status
- accconst
• useconst
• keywords/theme
• keywords/place
• keywords/stratum
• keywords/temporal

LEADresource/data/metainfo

Within LEADresource/data/geospatial:
• idinfo/timeperd
• idinfo/spdom/bounding
• idinfo/spdom/dsgpoly
• idinfo/vertdom

Within LEADresource/data/eainfo:
• detailed
• overview

Within LEADresource/data/distinfo:
• distrib/cntinfo
• stdorder

Within LEADresource/data/dataqual:
• complete
• lineage/procstep

As is the case with adding attributes, the item XML fragment for the update must be namespace qualified, but the namespace prefixes used in the fragment can differ from the prefixes used in the original document (and could also vary between fragments when using version of updateAttribute which takes an array of UpdateAttrHolder instances).

position:
Some metadata attributes allow for multiple instances, such as the theme attribute (the path to theme in the LMS is LEADresource/data/idinfo/keywords/theme). In cases such as updating a theme metadata attribute, the update must also specify which theme instance should be updated – the position parameter is used for this purpose. When a user queries the myLEAD metadata catalog, all instances of a specific metadata attribute (e.g., theme) are returned in the order they were added. If the user wishes to update the 3rd of 5 instances of the theme metadata attribute for an object, the position parameter should be set to 3 (the first instance is position 1, not 0). If there is more than one instance of a specific metadata attribute for an object, then any update of that metadata attribute must have a position parameter greater than zero or the update will not be performed. Likewise, if a position greater than the number of instances is specified as the position, then the update will not be performed.
When the results of a query are returned, the results are rooted in a `myQuery` tag and that tag contains an XML attribute named `timestamp` that contains the current date and time as a long integer in the following format: `YYYYMMDDHHMMSS` where the year in `YYYY` is 1970 or later. When a user specifies an update, this value should be returned as a `Long` in the `lastUpdate` parameter. Please note that the `updateAttribute` method takes a `Long` and not the native type `long`. The reason for this is to allow `null` to be passed for the `lastUpdate` parameter. A `null` value should only be passed for this parameter when the update is system generated – in that case there is no user looking at query results.

The second version of the `updateAttribute` method takes an array of `UpdateAttrHolder` instances – each of which contains `objectGuid`, `item`, `attributePos` (position), and `dataTimestamp` (lastUpdate) variables. This version of the method allows for multiple updates – possibly to different objects, but for a single user.

The `UpdateAttrHolder` class has two constructors:

```java
UpdateAttrHolder (String myItem, String guid, Integer position, Long myTimeStamp)
UpdateAttrHolder (String myItem, String guid)
```

If the second version is used, then both the `position` and `myTimeStamp` parameters default to `null`. Please note that in the first version of the constructor, these parameters use the `Integer` and `Long` classes respectively – not the primitive `int` and `long`. If `null` is passed for the `position` parameter, then there can be only one instance of the specified metadata attribute within the object being updated. If `null` is passed for the `myTimeStamp` parameter, then the myLEAD server cannot check whether the metadata attribute has been updated (and thus whether the `position` parameter setting may be out of date).

**Example: Updating Metadata Attributes**

```java
import edu.indiana.dde.mylead.client.*;
import edu.indiana.dde.mylead.common.*;
...
String dn = "/C=US/O=Town of Bedrock/CN=Fred Flinstone";
String guid = "urn:uuid:678580d4-6a48-1030-842e-000bda158a1";
long lastUpdate = 20060320111213; // 11:12am on the 20th of March, 2006
...
... <LeadClient created as client> ...
...
ReturnType ret = client.updateAttribute(dn, guid, 
"<MetadataAttribute/>",
null, New Long(lastUpdate) );
Client.close();
```

In the above example, the `position` parameter is set to `null`. Normally the `dn`, global ID, and last update timestamp would be passed in as parameters and not hard-coded. If the `position` were used, it would need to be passed as an `Integer` and not an `int`. 
The one object that a user cannot delete is the whiteboard default project. That project is needed to handle importing data, so deleting it would preclude importing any additional data from the public data catalog.

7 Deleting Metadata Objects

Metadata for objects in the myLEAD catalog can be deleted using the `delete` method:

```
delete (String dn, String objectGuid)
```

**Parameters:**
- **dn:**
  The `dn` parameter is the distinguished name of the user who is deleting the object (and must also be the owner of the object).

- **objectGuid:**
  This is the global ID (resourceID in the LMS) of the object for which the metadata is to be deleted. The object identified by this ID must already exist in myLEAD and the user identified by the `dn` parameter must own this object. If the object is a collection, experiment, or project then it may contain files, collections and possibly experiments (in the case where a project is being deleted). All of the metadata for the objects contained within the object being deleted are also deleted.

The Delete method deletes the metadata. It does not delete the actual data.

8 Deleting Metadata Attributes Within Objects

There are three versions of the `deleteAttribute` method, but the only difference between them is whether they take a single `position` parameter, an array of positions, or a range of positions:

```
deleteAttribute(String dn, String objectGuid, String attrName, String attrSrc, int position, Long lastUpdate)
deleteAttribute(String dn, String objectGuid, String attrName, String attrSrc, int[] positions, Long lastUpdate)
deleteAttribute(String dn, String objectGuid, String attrName, String attrSrc, int firstPos, int lastPos, Long lastUpdate)
```

If a file has six `theme` metadata attributes, a user could specify the `theme` metadata attribute and pass an array with two Integers (2, 4) and the 2\textsuperscript{nd} and 4\textsuperscript{th} instances of the `theme` metadata attribute for that file would be deleted. Alternately, the user could provide a value of 2 for the `firstPos` parameter and a value of 4 for the `lastPos` parameter. In that case, the 2\textsuperscript{nd}, 3\textsuperscript{rd}, and 4\textsuperscript{th} `theme` metadata attributes would be deleted.

**Parameters:**
- **dn:**
  The `dn` parameter is the distinguished name of the user who is deleting the metadata attribute (and must also be the owner of the object the metadata attribute relates to).
**objectGuid:**
This is the global ID (resourceID in the LMS) of the object for which the metadata attribute is being deleted. The object identified by this ID must already exist in myLEAD and the user identified by the dn parameter must own this object.

**attrName:**
Each metadata attribute defined in myLEAD is either based on a node in the LMS that has been identified as metadata attribute or is what we refer to as a dynamic metadata attribute. See the list above in the discussion regarding the updateAttribute method for a list of metadata attribute nodes in the LMS. Due to the LMS being a profile of the FGDC schema (which has an 8-character limitation on tag length), some of the tags have names that are not readily apparent – these have longer names in the myLEAD metadata catalog that are more descriptive. See Appendix A for a mapping from LMS nodes to myLEAD metadata attribute names.

The detailed node in the LMS is used to define those metadata attributes that are not defined based on the structure of the LMS – examples of such metadata attributes are notification messages, namelist parameters, and cross-cutting parameters (such as dx, dy, ctrlat, and ctrlon). Since there are no nodes in the LMS for ctrlat and ctrlon, a dynamic metadata attribute is defined for the cross-cutting parameters based on the detailed node in the LMS. When the detailed node is used to define a metadata attribute, the name of the attribute is the value in the enttyp node within the detailed node. It is the value in the enttyp node that would be passed for the attrName parameter in a call to the deleteAttribute method.

**attrSrc:**
Each metadata attribute defined in myLEAD has a name and a source, which when deleting a metadata attribute are the attrName and attrSrc parameters respectively. For all metadata attributes other than those defined based on the detailed node, the source is “LEAD”. It is because of the dynamic metadata attributes based on the detailed node in the LMS that the source parameter is needed. For example, metadata attributes with the same name could be used in two different models – such as ARPS and ADAM, or from two different namelist files. Each metadata attribute defined in myLEAD is unique based on the combination of its name and source. Each source is a string, but could be a URI identifying the model or other source of the metadata. Within the detailed node of the LMS, the source for a dynamic metadata attribute is the value in the enttypds node.

All of the metadata attributes in myLEAD are defined in the lead_attribute_definition table in the myLEAD database.

**position / positions:**
The position parameter is needed for those metadata attributes that could have multiple instances (e.g., the theme metadata attribute). See the discussion regarding the position parameter for the updateAttribute method. Unlike that method, the deleteAttribute method also allows for an array of positions since a user may
want to delete more than one instance of a particular metadata attribute (whereas for updates these would be separate update items).

**firstPos and lastPos:**
When a user wishes to delete a range of positions for a specific metadata attribute, then they can either specify each position in an array of positions or they can specify a `firstPos` and `lastPos` parameter. The first and last positions are inclusive – they will be part of the deletion.

**lastUpdate:**
Unlike the `updateAttribute` method, the `lastUpdate` parameter in the `deleteAttribute` method is required. The reason for this is that the `updateAttribute` method will sometimes be called directly by the workflow engine (possibly through the myLEAD Agent) without a user in the loop to update a metadata attribute – such as the status of the experiment. However, we do not foresee any reason that the workflow or myLEAD Agent would be deleting metadata without user intervention. If there is a user in the loop, then the `lastUpdate` parameter value is a check on whether the user is making the decision to delete a metadata attribute based on current metadata.

### Example: Deleting Metadata Attributes
```java
import edu.indiana.dde.mylead.client.*;
import edu.indiana.dde.mylead.common.*;
...
String dn = "/C=US/O=Town of Bedrock/CN=Fred Flinstone";
String guid = "urn:uuid:678580d4-6a48-1030-842e-000bda158a1";
long lastUpdate = 20060320111213; // 11:12am on the 20th of March, 2006
...
... <LeadClient created as client> ...
...
ReturnType ret = client.deleteAttribute (dn, guid, "theme", "LEAD", 3, 5, lastUpdate);
Client.close();
```

The above example illustrates deleting the 3rd through 5th instances of the theme keyword metadata attribute for an object. As in other examples, the dn, global ID, and last update timestamp would generally be passed in as parameters to a method instead of being hard-coded. As noted above, since the theme metadata attribute is based on the structure of the LMS, the `attrSource` parameter is “LEAD”.

## 9 Moving Metadata Objects in the Hierarchy

When objects are added to the myLEAD metadata catalog using the `create` method as described above, either a parent object’s global ID is provided or the object being added is a project (which never has a parent). However, there may be situations in which a user wishes to restructure the arrangement of their data (e.g., they decide to break out an experiment into a separate project). In that case the `move` method can be used to move one object (and any subtree of children underneath it), from one parent to a different parent. One case where this is used is when importing data into a user’s myLEAD space. When a user is created, a default project know as the “whiteboard” is created for them. Whenever data products are imported, these new data projects are automatically
assigned to the whiteboard and the user can later move them into another place in their hierarchy of projects, experiments, and collections. The move method is as follows:

```java
move (String dn, String objectGuid, String newParentGuid)
```

**Parameters:**

- **dn:**
  The dn parameter is the distinguished name of the user who is moving the object (they must also be the owner of the object being moved and the new parent it will be moved under).

- **objectGuid:**
  This is the global ID (resourceID in the LMS) of the object that is being moved. The object identified by this ID must already exist in myLEAD and the user identified by the dn parameter must own this object.

- **NewParentGuid:**
  This is the global ID (resourceID in the LMS) of the object that should be the new parent of the object being moved. As is the case for the objectGuid, the user identified by the dn parameter must own this object.

### 9.1 Determining a User’s Whiteboard Project

As mentioned above, when a new user is created in myLEAD, a new project called the “whiteboard” is defined for them and that project serves as the default destination when data products are imported to the user’s space. In order to create new objects within this project, the myLEAD Agent needs to be able to determine the global ID of the whiteboard project. To determine the global ID of a user’s whiteboard project, the getWhiteboardId method can be called:

```java
getWhiteboardId (String dn, LeadStringHolder results)
```

**Parameters:**

- **dn:**
  The dn parameter is the distinguished name of the user who’s whiteboard project must be determined.

- **results:**
  Query methods in the myLEAD metadata catalog are discussed in more detail in a later section. Each query will have a results parameter that is an instance of the LeadStringHolder class. This class is just a wrapper around a String variable named value that will contain the results – in this case the global ID of the user’s whiteboard project.
Example: Query for Whiteboard and Move Object to Whiteboard

```java
import edu.indiana.dde.mylead.client.*;
import edu.indiana.dde.mylead.common.*;
...
String dn = "/C=US/O=Town of Bedrock/CN=Fred Flinstone";
String guid = "urn:uuid:678580d4-6a48-1030-842e-000b6ba158a1";
...
<LeadClient created as client> ...

LeadStringHolder whiteboard = new LeadStringHolder (new String(""));
ReturnType ret = client.getWhiteboardId (dn, whiteboard);
if (!ret.equals (ReturnType.OPERATION_SUCCESSFUL)) {
    throw new MyLeadException ("Whiteboard ID could not be determined");
}
ReturnType ret = client.move (dn, guid, whiteboard.value);
Client.close();
```

The above example shows the `getWhiteboardId` method being used to get a user’s whiteboard and then moving an object onto the whiteboard (removing it as a child of its former parent if the object moved is not a project). As in the previous examples, the dn and global ID (resourceID) would usually be passed as parameters to a method and not hard-coded.

10 Querying the myLEAD Metadata Catalog

In myLEAD, there are three approaches to issuing a metadata query:

1. Context Queries
2. Query by Object ID
3. Query User Workspace

Context queries use the `queryLead` method of the myLEAD client API and search for objects based on metadata attributes, such as looking for all collections with “Katrina” in the title, covering a specific lat/lon point, and contained within a certain temporal range. In contrast, the query by ID uses the `queryById` method of the API and retrieves a specific object’s metadata based on the global ID (resourceID) passed as a parameter. Both the `queryLead` and `queryById` methods also allow the user to determine the extent of the metadata returned for each object in the result based on content filters, hierarchy filters, or exclusions. The filters and exclusion work the same for both types of queries and are covered in detail later.

The workspace query only returns the object type, global ID and title for each object and no filters or exclusions apply. The workspace query was designed to be efficient at returning only a limited and fixed set of data about each object.

10.1 Query Response Format

Both the `queryLead` and `queryById` methods have the same filter options and use the same response format - the query identifies the objects, and then the filters determine the portion of the LMS included in the response for each object. All of the query methods take an instance of the `LeadStringHolder` class as a parameter, and the query response is returned as a String containing an XML fragment in that parameter. The `LeadStringHolder` class has a public `String` variable named
value that contains the XML of the query response. In addition, that class has
\texttt{toString()} and \texttt{toXml()} methods that will return the query response XML as a
\texttt{String} or as an XML Bean Java class named \texttt{MyQueryDocument} respectively. If the
\texttt{LeadStringHolder} does not contain a query response (value is null) then the
\texttt{toString} method returns an empty \texttt{String}. If there is no query response or a
parsing error is encountered in the XML Bean class, then the \texttt{toXml} method returns
null. Please see the JavaDocs for the query response Bean and the LMS bean as to what
methods are available in the \texttt{MyQueryDocument} XML Bean class.

In the query response, to be consistent with the LMS (in which all elements are
qualified), the elements in the query response are also namespace qualified. Those
elements that are not from the FGDC schema, or one of the LEAD schemas, use the
MYLEAD namespace. The query result is always an XML fragment enclosed within
\texttt{<myQuery></myQuery>} tags. Within these tags, each object is enclosed in
\texttt{<project>, <experiment>, <collection>, or <file>} tags as applicable to the
object. The opening tag for each object contains an XML attribute named \texttt{resourceID}
with the global ID of the object as the attribute value. An example is the following
opening project tag:
\texttt{<ml:project resourceID="urn:uuid:678580d4-6a48-1030-842e-000bda158a1"}>

The contents within each object tag (e.g., \texttt{<project></project>}) is either a LMS
XML document or a subset of that schema based on the content filter that was specified
for the query. In addition, if the \texttt{SUBTREE} filter is used, child objects are included
within the parent object tags.

### Example: Query Response XML

```
<ml:myQuery xmlns:ml="MYLEAD" timestamp=20061029211300">
  <ml:project resourceID="urn:uuid:678580d4-6a48-1030-842e-000bda158a1">
      ...
    </lead:LEADresource>
  </ml:project>
  <ml:experiment resourceID="urn:uuid:678580a2-6a48-1030-842e-000bda158a1">
      ...
    </lead:LEADresource>
  </ml:experiment>
  <ml:collection resourceID="urn:uuid:678580b3-6a48-1030-842e-000bda158a1">
      ...
    </lead:LEADresource>
  </ml:collection>
</ml:myQuery>
```

### 10.2 Context Queries

This type of query provides the ability to select objects based on metadata properties
(e.g., looking for collections with “Katrina” in the title, covering a specific lat/lon point,
and a certain temporal range). In addition, the query could specify a context for an
object such as specifying that the query is targeting collections that must contain files
with specific theme keywords. The \texttt{queryLead} method in the \texttt{LeadClient} class is
used to issue context queries. All variations of the \texttt{queryLead} method take a target
parameter of type MyLeadQuery. This specifies the object(s) being searched for. In addition, some variations of this method also take a parameter named query that is of type MyLeadContextQuery. The query parameter specifies the context in which the target must be contained. Using the example mentioned above, the target may be collections with “Katrina” in the title, covering a specific lat/lon point, and a certain temporal range, whereas the context specified in the query parameter may be that the collections must contain files with certain theme keyword metadata attributes. If a context is specified, then objects matching the target must also match the query context in order to be included in the results returned. After myLEAD has determined which target objects also match the context, the hierarchy filter (discussed later) is applied to those targets to determine the hierarchy returned. The MyLeadContextQuery context and the hierarchy filter are independent – the MyLeadContextQuery is used to determine which targets meet the query criteria, and the hierarchy filter determines the context (parent and child objects) included with the target(s) in the results returned.

There are eight versions of the queryLead method that allow for various combinations of hierarchy and content filters, plus four versions that also allow for the exclusion of selected metadata attributes:

queryLead(String dn, int limit, String hFilter, String cFilter, MyLeadQuery target, MyLeadContextQuery query, LeadStringHolder results)

queryLead(String dn, int limit, MyLeadParentChildFilterSet hFilter, String cFilter, MyLeadQuery target, MyLeadContextQuery query, LeadStringHolder results)

queryLead(String dn, int limit, String hFilter, MyLeadContentFilter cFilter, MyLeadQuery target, MyLeadContextQuery query, LeadStringHolder results)

queryLead(String dn, int limit, MyLeadParentChildFilterSet hFilter, MyLeadContentFilter cFilter, MyLeadQuery target, MyLeadContextQuery query, LeadStringHolder results)

queryLead(String dn, int limit, String hFilter, String cFilter, MyLeadQuery target, LeadStringHolder results)

queryLead(String dn, int limit, MyLeadParentChildFilterSet hFilter, String cFilter, MyLeadQuery target, LeadStringHolder results)

queryLead(String dn, int limit, String hFilter, MyLeadContentFilter cFilter, MyLeadQuery target, LeadStringHolder results)

queryLead(String dn, int limit, MyLeadParentChildFilterSet hFilter, String cFilter, MyLeadQuery target, LeadStringHolder results)

Methods that allow nodes or attributes to be excluded:

queryLead(String dn, int limit, String hFilter, String cFilter, QryExclusionHolder[] exclusions, MyLeadQuery target, MyLeadContextQuery query, LeadStringHolder results)

queryLead(String dn, int limit, MyLeadParentChildFilterSet hFilter, String cFilter, QryExclusionHolder[] exclusions, MyLeadQuery target, MyLeadContextQuery query, LeadStringHolder results)

queryLead(String dn, int limit, String hFilter, String cFilter, QryExclusionHolder[] exclusions, MyLeadQuery target, LeadStringHolder results)

queryLead(String dn, int limit, MyLeadParentChildFilterSet hFilter, String cFilter, QryExclusionHolder[] exclusions, MyLeadQuery target, LeadStringHolder results)
**Parameters:**

**dn:**
The `dn` parameter is the distinguished name of the user who’s issuing the query and who’s workspace will be searched. Even if the query criteria are matched by objects in another user’s workspace, those objects are not shown.

**limit:**
This is an `int` that specifies if the number of target objects returned should be limited. Passing zero for this parameter indicates no limit. For example, if the query targets experiments and the `limit` is set to five, then at most five experiments will be returned. Depending on the hierarchy filter used (see below), there may also be parent and child objects returned too. The limit is applied before the hierarchy filter.

**hFilter:**
This filter determines the hierarchy returned for each object that meets the criteria for the query’s target. This filter can be either a simple filter or a more detailed parent-child filter. The three options for the simple version of this filter are: `TARGET`, `SUBTREE`, and `CHILDREN`. See the discussion below for details on the hierarchy filter.

**cFilter:**
This filter is used to determine what results should be included within each object returned based on the hierarchy filter. The content filter options are:

- GUIDONLY
- FULL_SCHEMA
- Selected Attributes
- Selected Elements

See the discussion below for details on the content filter.

**target:**
This parameter specifies the metadata attribute and metadata element criteria for the target of the query. The target is an instance of a class that extends the `MyLeadQuery` class. The classes that extend `MyLeadQuery` are based on the data hierarchy used in myLEAD: `MyLeadFile`, `MyLeadCollection`, `MyLeadExperiment`, and `MyLeadProject`. Each of these extensions only adds its object type as a flag when the query parameters are wrapped into the perform document for the query. All of these extensions have the same capabilities based on the `MyLeadQuery` class. This class is discussed in detail below in the section on defining metadata attribute and element criteria for a query.

**query:**
This parameter specifies the context that the target must be contained within. If the target of our query is collections, but we want to also specify that the collections must contain files with certain properties, and that the collections must be contained within experiments that have certain properties. In that case, the file and experiment criteria make up the context to be included in this parameter. This query context, similar to the target itself, uses the `MyLeadQuery` class to describe each object in the context. This
parameter is discussed in detail below in the section on defining metadata attribute and element criteria for a query.

**exclusions:**
This parameter can be used to indicate metadata attributes that should be excluded from the results returned when the FULL_SCHEMA option is used for the content filter. However, the exclusions cannot result in a document that would not validate against the LMS or that particular exclusion will be ignored. See the discussion below regarding exclusions.

**results:**
Each query in myLEAD will have a results parameter that is an instance of the LeadStringHolder class. This class is just a wrapper around a String variable named value that will contain the results. For all context queries the result will be wrapped in a set of myQuery tags and the opening tag will have an XML attribute named timestamp with the date/time on the server when the answer was current. It is this timestamp that should be returned as a parameter in calls to the updateAttribute or deleteAttribute methods as discussed above. The LeadStringHolder class also has a toXml method that can be used to get the query result as an XML Bean instance of the class MyQueryDocument.

### 10.2.1 Defining Metadata Attribute and Element Criteria For a Query
Both the target of a query and the context of a query are based on the MyLeadQuery class. This class only has a default constructor, but there are methods in the class to set query criteria based on when the object was created as well as metadata attributes of the object. When an object is initially added in myLEAD, its creation date is set. The MyLeadQuery class has the following methods that allow criteria for the creation date to be specified:

```java
void creatorSearchRng (Date startDate, Date endDate)
void setStartCreateRng (Date startDate)
void setEndCreateRng (Date endDate)
```

If either the starting or ending date is null (the default in the constructor) then that criteria does not apply. For example, if only the setEndCreateRng method is called, then the target can have been created any time up to the date set in the call to that method. There are also getter methods `getStartCreateRng` and `getEndCreateRng`.

**Metadata Attribute Query Conditions:**
There is no requirement that the target contain any such criteria - the user could be querying for all collections containing files with some specific characteristics. In that case there would be no criteria on the collection target. Any metadata attribute criteria specified provide a minimum condition for an object to match the query. The method used to add metadata attribute criteria to the query is the `addAttribute` method:
void addAttribute (MyLeadAttribute leadAttribute)

Metadata attributes are defined in myLEAD as containing metadata elements and possibly sub-attributes. In addition to a default constructor, the MyLeadAttribute class has two additional constructors:

MyLeadAttribute (String attrName)
MyLeadAttribute (String attrName, String attrSrc)

As discussed in section 8 on deleting metadata attributes, each defined metadata attribute has a name and a source. If an instance of MyLeadAttribute is added to the MyLeadQuery used for the query’s target without having added any metadata element criteria, the query will only check for the existence of the specified attribute – regardless of the metadata element values it contains. The addElement method in MyLeadAttribute is used to specify element conditions on the attributes and it only takes one parameter, the metadata element criteria:

void addElement (MyLeadElement leadElement)

If a metadata attribute is defined in myLEAD as having multiple elements, the query does not need to include all of the defined elements – the user can optionally specify only the elements that are critical to their search. For the elements, there are constructors for matching strings, values, or a range of values:

MyLeadElement(String name, String elementSrc, String attrValue, boolean exact)
MyLeadElement(String name, String elementSrc, String attrValue, int compare)
MyLeadElement(String name, String elementSrc, String attrStart, String attrEnd)
MyLeadElement(String name, String elementSrc, double xCoordinate, double yCoordinate)

MyLeadElement Constructor Parameters:

name:
This is the name of the metadata element as defined in myLEAD. Definitions for all metadata elements in myLEAD can be found in the lead_element_definition table in the myLEAD database. For dynamic metadata elements based on the detailed node of the LMS, the value of the arrtlbl node is the name of the metadata element.

elementSrc:
Each metadata element defined in myLEAD is unique based on a combination of its name and source. This parameter provides the source. As with metadata attributes, all of the metadata elements defined based on the LMS (not within a detailed node) have a source of “LEAD”. For metadata elements based on the detailed node, the source is the value of the attrdefs node.

attrValue:
Although this parameter is a string, it should be representative of the data type for the specific element. In performing the comparison, myLEAD tries to convert the string to the relevant data type (e.g., date, time, integer, float). If the value for any criteria cannot
be converted, then that element is disregarded for the query. The data type for each metadata element is specified in the lead_element_definition table in the myLEAD database.

**exact:**
The first version of the constructor should only be used with metadata elements that are of type String. This parameter is then used to indicate if the contents of the metadata element need to be an exact match with the value specified (true) or only need to contain the value (false).

**compare:**
This parameter is one of the following constants defined in the LeadConstants class:
- MYLEAD_EQUAL = 0
- MYLEAD_NOT_EQUAL = 1
- MYLEAD_GREATER_THAN = 2
- MYLEAD_GREATER_THAN_EQUAL = 3
- MYLEAD_LESS_THAN = 4
- MYLEAD_LESS_THAN_EQUAL = 5
- MYLEAD_CONTAINS = 3
- MYLEAD_WITHIN = 5
- MYLEAD_INTERSECTS = 6
- MYLEAD_NOT_INTERSECTS = 7

Along with MYLEAD_EQUAL and MYLEAD_NOT_EQUAL, the last four options apply to spatial comparisons. The “contains” and “greater than or equal” constants have the same value, but the contains constant makes more sense from a user’s perspective for spatial comparisons. Likewise for the “within” and “less than or equal” constants. When the MYLEAD_CONTAINS comparison is used, the query will search for spatial values in the specified metadata element that contain the spatial value specified in the query, so the data value must be equal or larger than the value specified in the query. For the MYLEAD_WITHIN comparison it is the opposite – the metadata element value must be within the spatial bounds specified in the query.

Currently spatial comparisons within myLEAD are performed by comparing the bounding boxes of the data and the value specified in the query. A query can specify a point or a polygon. Currently the only spatial metadata being stored in myLEAD is the spatial metadata element boundingBox, which is within the metadata attribute spatialBounds, which is constructed from the coordinates specified within the bounding node of the LMS.

**AttrStart and attrEnd:**
These parameters are Strings, but they should be representative of the data type for the specific element. In performing the comparison, myLEAD tries to convert the string to the relevant data type (e.g., date, time, integer, float). If the value cannot be converted, then the element is discarded as a criteria for the query.
**xCoordinate and yCoordinate:**
The fourth constructor shown above provides the ability to do a spatial comparison for any spatial metadata element defined in myLEAD. These two parameters specify a point. This sets the type of comparison to a point. Additional points can be added to the metadata element using the addPoint method to instead make the comparison be a polygon. However, if additional points are added, there needs to be at least a total of three points for the polygon.

The **addPoint** method is as follows:

```java
void addPoint (double xCoordinate, double yCoordinate)
```

**Example: Using Metadata Attributes and Elements in Queries**

```java
import edu.indiana.dde.mylead.client.*;
import edu.indiana.dde.mylead.common.*;
...
String dn = "/C=US/O=Town of Bedrock/CN=Fred Flinstone";
...
...<LeadClient created as client> ...
...
MyLeadQuery myObject = new myExperiment();
MyLeadAttribute attribute = new MyLeadAttribute("citation","LEAD");
attribute.addElement(new MyLeadElement("title", "LEAD", "Katrina", false));
myObject.addAttribute(attribute);
LeadStringHolder result = new LeadStringHolder (new String(""));
ReturnType ret = client.queryLead(dn, 0, "TARGET", "FULL_SCHEMA", myObject, result);
client.close();
return (result.value); //returns the query results
```

This example queries for experiments that contain the string “Katrina” anywhere in the title and returns the full metadata (FULL_SCHEMA) for the experiments meeting the criteria, but not their child objects (since the TARGET context filter is used). The FULL_SCHEMA and TARGET are content and hierarchy filters respectively and are discussed in detail after the section on querying based on object IDs. For this example the dn, metadata attribute and element names and sources, the value being searched for, and the boolean flag as to whether the match should be exact were all hard-coded, but these should be method parameters. Likewise, we did not check the value of the ReturnType returned by the call to queryLead. This should be ReturnType.OPERATION_SUCCESSFUL if the call was successful.

### 10.3 Query By Object ID

This type of query should be used when you are looking for the metadata related to a specific object or the context of a specific object. The **queryById** method in the LeadClient class is used for these queries. There are eight versions of the queryById method that allow for various versions of the hierarchy and content filters, plus four versions that also allow for exclusions:

```java
queryById(String dn, String hFilter, String cFilter, String[] guids, LeadStringHolder results)
queryById(String dn, MyLeadParentChildFilterSet hFilter, String cFilter, String[] guids, LeadStringHolder results)
```
queryById(String dn, String hFilter, MyLeadContentFilter cFilter, String[] guids, LeadStringHolder results)

queryById(String dn, MyLeadParentChildFilterSet hFilter, MyLeadContentFilter cFilter, String[] guids, LeadStringHolder results)

queryById(String dn, String hFilter, String cFilter, String guid, LeadStringHolder results)

queryById(String dn, MyLeadParentChildFilterSet hFilter, String cFilter, String guid, LeadStringHolder results)

queryById(String dn, String hFilter, MyLeadContentFilter cFilter, String guid, LeadStringHolder results)

queryById(String dn, MyLeadParentChildFilterSet hFilter, MyLeadContentFilter cFilter, String guid, LeadStringHolder results)

queryById(String dn, String hFilter, MyLeadContentFilter cFilter, String[] guids, LeadStringHolder results)

queryById(String dn, MyLeadParentChildFilterSet hFilter, MyLeadContentFilter cFilter, String[] guids, LeadStringHolder results)

queryById(String dn, String hFilter, String cFilter, QryExclusionHolder[] exclusions, String[] guids, LeadStringHolder results)

queryById(String dn, MyLeadParentChildFilterSet hFilter, String cFilter, QryExclusionHolder[] exclusions, String[] guids, LeadStringHolder results)

queryById(String dn, String hFilter, String cFilter, QryExclusionHolder[] exclusions, String guid, LeadStringHolder results)

queryById(String dn, MyLeadParentChildFilterSet hFilter, String cFilter, QryExclusionHolder[] exclusions, String guid, LeadStringHolder results)

Methods that allow nodes or attributes to be excluded:

queryById(String dn, String hFilter, String cFilter, QryExclusionHolder[] exclusions, String[] guids, LeadStringHolder results)

queryById(String dn, MyLeadParentChildFilterSet hFilter, String cFilter, QryExclusionHolder[] exclusions, String[] guids, LeadStringHolder results)

Parameters:

dn:
The dn parameter is the distinguished name of the user who’s issuing the query and who’s workspace will be searched. Even if the query criteria are matched by objects in another user’s workspace, those objects are not shown.

hFilter:
This filter has the same options as the hFilter option for queryLead.

cFilter:
This filter has the same options as the cFilter option for queryLead.

guid / guids:
This is the global ID (or array of global IDs) for the object(s) to be returned by the query. In comparison to the queryLead method, this parameter sets the targets and then the hierarchy and content filters determine the context and content for the guid(s) provided in this parameter.

exclusions:
This parameter can be used to indicate metadata attributes that should be excluded from the results returned when the FULL_SCHEMA option is used for the content filter. However, the exclusions cannot result in a document that would not validate against the LMS or that particular exclusion will be ignored. See the discussion below regarding exclusions.
Each query in myLEAD will have a results parameter that is an instance of the LeadStringHolder class. See the discussion of the results parameter for the queryLead method as discussed above.

**Example: Querying by ID**

```java
import edu.indiana.dde.mylead.client.*;
import edu.indiana.dde.mylead.common.*;
...
String dn = "/C=US/O=Town of Bedrock/CN=Fred Flinstone";
String guid = "urn:uuid:678580d4-6a48-1030-842e-000bd8a158a1";
...
... <LeadClient created as client> ...
...
LeadStringHolder result = new LeadStringHolder (new String(""));
ReturnType ret = client.queryById(dn, "SUBTREE", "FULL_SCHEMA", guid, result);
client.close();
return (result.value); //returns the query results
```

This example would return all of the metadata for the object identified by the specified global ID, including all of the children of that object (e.g., if the guid variable represents an experiment, it would include all child collections and files, including the files within the collections).

### 10.4 Query Filters

There are two types of filters available: hierarchy filters and content filters. Both filters are applied only after the objects matching the query are determined. In the case of the queryLead method, these objects are determined based on the target parameter and optionally the query context. In the case of the queryById method, these objects are specified by the guid parameter. After the objects matching the query are determined, the hierarchy filter is applied to determine the context to be included with the result, then the content filter determines what should be returned for each object that is included in the result (including the initial query results and the context specified by the hierarchy filter).

#### 10.4.1 Hierarchy Filters

This filter has two flavors, the first is a string parameter equals TARGET, SUBTREE, or CHILDREN. The TARGET filter returns data for only those objects matching the target of a context query or the guid specified in the queryById method. The SUBTREE filter returns not only the target objects, but also the subtree of all child objects below each target object. In the query response, these child objects are nested within the parent objects. The CHILDREN filter differs in that it does not actually include the objects that match the query in the result, but instead returns only their direct children. One issue to note is that this filter currently works only with the first object identified by the query. The reason for this limitation is that it prevents the result from mixing the children of multiple objects in a single response. Since only the first target is used when the CHILDREN filter is set, the limit parameter of the query method has no effect since it places a limit on the number of targets to consider (and a value of zero is no limit).
The second flavor of the hierarchy filter is the `MyLeadParentChildFilterSet`. This filter allows the user to specify types of parent and child objects that should be included in the result. The constructor for the filter set class takes two boolean parameters:

```
MyLeadParentChildFilterSet (boolean parents, boolean children)
```

If the `parents` parameter is true, then the result will include parents of the target objects. Which parent objects will be included in the results can be tailored by setting a distance limit and/or parent types to be excluded. Likewise, the `children` parameter will determine if any children of the target should be included in the result. If both parameters are set to false, then the user will receive the same results as if the TARGET hierarchy filter had been used. Likewise, if the `parents` parameter is set to false, the `children` parameter is set to true, and no further options are set, then the result would be the same as if the SUBTREE hierarchy option had been used.

The filter set allows for setting the number of levels of parents or children should be included for each target included in the query results with the following methods:

```
setParentLimit (int distLimit)
setChildLimit (int distLimit)
```

If the `setParentLimit` method is called with a `distLimit` of 2, then the parent and grandparent of each object will be included in the result (to the extent that each target has 2 parents – a project has no parents).

In addition to specifying the number of levels of parents to include, the filter set also allows the user to specify object types that should be excluded from the parents and/or children included in the result. The following two methods can be called to set the object types to be excluded:

```
excludeParent (int parentType)
excludeChild (int childType)
```

The parameters for each call can be any of the following:

```
LeadConstants.MYLEAD_FILE
LeadConstants.MYLEAD_COLLECTION
LeadConstants.MYLEAD_EXPERIMENT
LeadConstants.MYLEAD_PROJECT
```

If an invalid exclusion type is set for the parent or the child (e.g., exclude collections as parents of experiments), it will be ignored in processing the query. Each exclude method can be called multiple times (e.g., if a user is querying for files and sets both the collection and experiment levels to be excluded, the result will include the parent projects and the files without any intervening layers).

If the user is querying for files, the `parents` parameter in the constructor was set to true, and the `excludeParent` method was called twice, once with the constant for
collections, and again with the constant for experiments, then the result returned would be hierarchies consisting of the project parent and the target files as their direct children.

In myLEAD, collections can contain nested collections. If the excludeParent method is called with LeadConstants.MYLEAD_COLLECTION as the type when the target of the query is collections, the target collections will be included in the result but any parent collections will be excluded (unless they also match the query criteria as targets). The same applies when the excludeChild method is used with collections – any child collections would be excluded.

Example: MyLeadParentChildFilterSet

```java
import edu.indiana.dde.mylead.client.*;
import edu.indiana.dde.mylead.common.*;
...
String dn = "/C=US/O=Town of Bedrock/CN=Fred Flinstone";
String guid = "urn:uuid:678580d4-6a48-1030-842e-000bda158a1";
...
MyLeadParentChildFilterSet filterSet = new MyLeadParentChildFilterSet (true, false);
filterSet.excludeParent (LeadConstants.MYLEAD_PROJECT);
filterSet.excludeParent (LeadConstants.MYLEAD_COLLECTION);
...
...<LeadClient created as client> ...
...
LeadvStringHolder result = new LeadStringHolder (new String(""));
ReturnType ret = client.queryById(dn, filterSet, "FULL_SCHEMA", guid, result);
client.close();
return (result.value); //returns the query results
```

The above example builds on the queryById example shown earlier. Instead of the SUBTREE hierarchy filter used earlier, a MyLeadParentChildFilterSet is created which will include parent objects (since the first parameter of the constructor is true), will exclude any child objects (since the second parameter of the constructor is false), and will exclude any project or collection parents.

10.4.2 Content Filters

These filters are used to determine what results should be included within each object returned based on the hierarchy filter. The content filter options are:

- GUIDONLY
- FULL_SCHEMA
- Selected Attributes
- Selected Elements

**GUIDONLY Content Filter:**

Since the global ID is included in the object tags, the GUIDONLY filter excludes any content within an object's opening and closing tags (other than nested objects).
Example: GUIDONLY Content Filter

```java
import edu.indiana.dde.mylead.client.*;
import edu.indiana.dde.mylead.common.*;

... String dn = "/C=US/O=Town of Bedrock/CN=Fred Flinstone";
String[] guids = {"urn:uuid:678580d4-6a48-1030-842e-000bdba158al",
                  "urn:uuid:678580a2-6a48-1030-842e-000bdca158a4"};

... <LeadClient created as client> ...

LeadStringHolder result = new LeadStringHolder (new String(""));
ReturnType ret = client.queryById(dn, "SUBTREE", "GUIDONLY", guids, result);
client.close();
return (result.value); //returns the query results
```

This example queries by ID, passing an array containing two global IDs. Assuming the two IDs passed were collections and each collection contained two files, the result of the query would be as follows:

```xml
<ml:myQuery xmlns:ml="MYLEAD">
  <ml:collection resourceID="urn:uuid:678580d4-6a48-1030-842e-000bdba158a1">
    <ml:file resourceID="urn:uuid:678571e1-6a48-1030-842e-000bdba158a1"/>
    <ml:file resourceID="urn:uuid:678571e2-6a48-1030-842e-000bdba158a1"/>
  </ml:collection>
  <ml:collection resourceID="urn:uuid:678580a2-6a48-1030-842e-000bdca158a4">
    <ml:file resourceID="urn:uuid:678570b8-6a48-1030-842e-000bdca158a4"/>
    <ml:file resourceID="urn:uuid:678570b9-6a48-1030-842e-000bdca158a4"/>
  </ml:collection>
</ml:myQuery>
```

**FULL_SCHEMA Content Filter:**

The FULL_SCHEMA returns the entire metadata document starting with a LEADResource element, including all namespaces and namespace prefixes that are applicable. If the SUBTREE hierarchy filter is applied, the result returned will include the full metadata document for all children below the object too. However, the FULL_SCHEMA filter option also allows the user to specify particular nodes in the LMS or particular metadata attributes that they wish to exclude from the result – so long as the exclusion would not result in a result that does not validate against the LEAD schema. See the section below on Query Exclusions.

Example: FULL_SCHEMA Content Filter

```java
import edu.indiana.dde.mylead.client.*;
import edu.indiana.dde.mylead.common.*;

... String dn = "/C=US/O=Town of Bedrock/CN=Fred Flinstone";
String[] guid = "urn:uuid:678580d4-6a48-1030-842e-000bdba158a1";

... <LeadClient created as client> ...

LeadStringHolder result = new LeadStringHolder (new String(""));
ReturnType ret = client.queryById(dn, "TARGET", "FULL_SCHEMA", guid, result);
client.close();
return (result.value); //returns the query results
```

In this example we are only querying for a single object based on its ID and using the TARGET hierarchy filter so we do not get all of the children included in the result. Since
the FULL_SCHEMA content filter includes all of the metadata for each object in the result, the query result can be large.

**Selected Attributes and Elements:**
While myLEAD is designed to communicate via LMS, there are times when a user or process may only need certain metadata attributes or metadata elements. There are two available filter classes—MyLeadAttributeFilter and MyLeadElementFilter. Both of these filters can be used as the content filter for either the queryLead or queryById methods in LeadClient.

**MyLeadAttributeFilter:**
The MyLeadAttributeFilter class takes the metadata attributed name and source as defined in myLEAD. This includes all of the blue, non-italicized nodes in the LMS schema diagram of Appendix B as well as any dynamic metadata attributes (contained in the “detail” nodes under geospatial in the LMS). The subtree from the LMS for the selected metadata attributes will be included in the response document, nested within each object's tags, in the order the attribute definitions were added to the instance of the MyLeadAttributeFilter class that was passed as a query parameter. As with the FULL_SCHEMA filter, namespaces and prefixes will be included for each metadata attribute based on the LMS. This filter class is defined in the edu.indiana.dde.mylead.common package. There are two versions of the constructor for this class in addition to the default constructor:

MyLeadAttributeFilter (String source, String name)
MyLeadAttributeFilter (String name)

When the source is not specified, it defaults to “LEAD” which is the source for all of the metadata attributes based on nodes other than the detailed node in the LMS. For the detailed node in the schema, the source of the metadata attribute is specified by the value in the enttyp node and the name is specified by the value in the enttypds node.

To have the filter include more than one metadata attribute in the result, the addAttribute method can be used:

void addAttribute (String source, String name)
Example: MyLeadAttributeFilter Content Filter

```java
import edu.indiana.dde.mylead.client.*;
import edu.indiana.dde.mylead.common.*;
...
String dn = "/C=US/O=Town of Bedrock/CN=Fred Flinstone";
String guid = "urn:uuid:678580d4-6a48-1030-842e-000bdba158a1";
...
... <LeadClient created as client> ...
...
MyLeadAttributeFilter myFilter = new MyLeadAttributeFilter("LEAD", "citation");
myFilter.addAttribute("LEAD", "status");
... 
LeadStringHolder result = new LeadStringHolder (new String(""));
ReturnType ret = client.queryById(dn, "SUBTREE", myFilter, guid, result);
client.close();
return (result.value); //returns the query results
```

This example queries by ID, but it filters the result so that only the metadata attributes 
`citation` and `status` are returned. The results of this query would have the 
following format:

```xml
<ml:myQuery xmlns:ml="MYLEAD">
  <ml:project resourceID="urn:uuid:678580d4-6a48-1030-842e-000bdba158a1">
    <lead:citation
      xmlns:lead="http://schemas.leadproject.org/2007/01/lms/lead"
      xmlns:fgdc="http://schemas.leadproject.org/2007/01/lms/fgdc">
      <fgdc:origin>Yiming Sun</fgdc:origin>
      <fgdc:origin>Yogesh Simmhan</fgdc:origin>
      <fgdc:pubdate>Unknown</fgdc:pubdate>
      <fgdc:title>Hurricane Wilma</fgdc:title>
      <fgdc:pubinfo>
        <fgdc:pubplace>Champaign Illinois</fgdc:pubplace>
        <fgdc:publish>LEAD Resource Catalog</fgdc:publish>
      </fgdc:pubinfo>
      <fgdc:othercit>other stuff</fgdc:othercit>
    </lead:citation>
    <lead:status
      xmlns:lead="http://schemas.leadproject.org/2007/01/lms/lead"
      xmlns:fgdc="http://schemas.leadproject.org/2007/01/lms/fgdc">
      <fgdc:progress>Complete</fgdc:progress>
      <fgdc:update>Unknown</fgdc:update>
    </lead:status>
  </ml:project>
</ml:myQuery>
```

MyLeadElementFilter:
The MyLeadElementFilter allows the user to select any number of metadata 
elements from the LMS – this requires the metadata attribute name and source as well as 
the name and source of the metadata element within that attribute. This filter also takes a "name" parameter for each metadata element specified in the filter. When the filtered metadata elements are included in the response document, each is wrapped in an "Element" tag with the specified name as an XML attribute. If no name is specified, the metadata element’s name (as defined in the lead_metadata_elements table in 
myLEAD will be used as the name). The reason for this approach is that the same name 
may be used for metadata elements within different metadata attributes within the LMS, 
so just using the LMS tag (even with namespaces) could be ambiguous.
This filter class is defined in the package as the attribute filter class. As with the attribute filter, there are two versions of the constructor for this filter in addition to the default constructor as well as an addElement method to add additional metadata elements to the filter:

```java
MyLeadElementFilter (String attrSource, String attrName, String elementSource, String elementName, String name)
MyLeadElementFilter (String attrName, String elementName, String name)
void addElement (String attrSource, String attrName, String elementSource, String elementName, String name)
```

**Example: MyLeadElementFilter Content Filter**

```java
import edu.indiana.dde.mylead.client.*;
import edu.indiana.dde.mylead.common.*;
...
String dn = "/C=US/O=Town of Bedrock/CN=Fred Flinstone";
String guid = "urn:uuid:678580d4-6a48-1030-842e-000bdba158a1";
...
... <LeadClient created as client> ...
...
MyLeadElementFilter myFilter = new MyLeadElementFilter("citation", "title", "title");
myFilter.addElement("LEAD", "keywordTheme", "LEAD", "themeKeyword", "theme");
...
LeadStringHolder result = new LeadStringHolder (new String(""));
ReturnType ret = client.queryById(dn, "SUBTREE", myFilter, guid, result);
client.close();
return (result.value); //returns the query results
```

This example queries by ID, but it filters the result so that only the title metadata element from the metadata attribute citation and the themeKeyword metadata element from the keywordTheme metadata attribute are returned. The results of this query would have the following format:

```xml
<ml:myQuery xmlns:ml="MYLEAD">
  <ml:project resourceID="urn:uuid:678580d4-6a48-1030-842e-000bdba158a1">
    <myLeadElement name="title">Hurricane Wilma</myLeadElement>
    <myLeadElement name="themeKeyword">geostrophic_eastward_wind</myLeadElement>
    <myLeadElement name="themeKeyword">surface_geopotential</myLeadElement>
    <myLeadElement name="themeKeyword">wind_mixing_energy_flux_into_ocean</myLeadElement>
  </ml:project>
</ml:myQuery>
```

As with the attribute filter, the elements are included within each object in the order they were specified in the filter – not their order in the schema.

**10.5 Query Exclusions**

When using the FULL_SCHEMA content filter option, the user can optionally exclude nodes from the LMS or specific metadata attributes by populating an array of
QryExclusionHolder instances as a parameter to either the queryById or queryLead method. Each instance of QryExclusionHolder has just two values:

- Metadata attribute or schema node name
- Source of the metadata attribute (not needed for schema nodes)

The constructors for a QryExclusionHolder object are:

QryExclusionHolder (String name, String source)
QryExclusionHolder (String name)

If no source is supplied, the instance is assumed to be a schema node tag and the source “LEAD” is used. The ability to exclude metadata attributes is used mainly to exclude metadata attributes stored within the detailed node in LMS. For example, if a user wants to exclude distinfo nodes and also loginfo type notifications (which are a defined metadata attribute within myLEAD based on the detailed node) they could use the code in the following example.

**Example: Attribute Exclusion Filter**

```java
QryExclusionHolder[] exclusions = new QryExclusionHolder[2];
exclusions[0] = new QryExclusionHolder("distinfo");
exclusions[0] = new QryExclusionHolder("logInfo", "http://lead.extreme.indiana.edu/namespaces/2006/06/workflow_tracking.logInfo");
```

The exclusions object in this example could then be included as the content filter parameter in a query.

If any of the nodes or metadata attributes to be excluded would result in a query response that would not validate against the LMS, then those particular exclusions will be ignored and any valid exclusions would still be done. For example, if the user included a QryExclusionHolder instance with the name set to “status” (the status node in LMS) then that exclusion would be ignored since status is a required node in the LMS.

**Excludable LEAD Metadata Schema Nodes:**

Following is a list of the LMS nodes that can be excluded:

Within LEADresource/data/idinfo/keywords:
- place
- stratum
- temporal

Within LEADresource/data/geospatial:
- idinfo/spdom/dsgpoly
- idinfo/vertdom

Within LEADresource/data/eainfo:
- eainfo/detailed
- eainfo/overview
LEADresource/data/distinfo
LEADresource/data/distinfo/stdorder
LEADresource/data/dataqual
LEADresource/data/enclosedresources

In addition, any metadata attribute that is defined to use the detailed node in the LMS can also be excluded. These include the namelist parameter blocks, cross-cutting attributes, and notifications.

There are other nodes in the LMS that are optional, but these cannot be excluded when returning query results for one of the following two reasons:

1. **Nodes that are within a metadata attribute:**
   An example is the `pubtime` node that is within the `citation` node. Each metadata document is stored in chunks (CLOBs) in myLEAD based on nodes in the LMS that have been identified as metadata attributes. The `citation` node is defined as a metadata attribute in myLEAD, so a query cannot exclude a node that is within a `citation` node without excluding the entire node. In this case, the `citation` node cannot be excluded because it is mandatory in the LMS.

2. **Nodes that would result in an invalid LMS document:**
   Some nodes within optional nodes are required if their optional parent is included. Two examples in the LMS are the `idinfo` node within the `geospatial` node and the `distrib` node within the `distinfo` node. In the case of the `distrib` node, there could be optional `stdorder` nodes that are also children of the `distinfo` node. If the `distrib` node was allowed to be excluded, then a document could be returned that had a `distinfo` node with only `stdorder` child nodes. Such a document would not validate against the LMS. In such a case, if a user knows there are no `stdorder` nodes, they could exclude the `distinfo` node (which is allowed to be excluded).
Appendix A: Metadata Attribute and Element Names

Since the FGDC schema limits the short names to 8 characters, the tags in the FGDC (and LEAD) schemas can be difficult at times to decipher. In the attribute and element definition tables, some of the attributes and elements were given longer names. The names to be specified in the attribute and element content filters are then names used in the definition tables in myLEAD.

The majority of the metadata attributes and elements defined in myLEAD use the detailed node in the schema. We call these dynamic metadata attributes, and their name and source are based on the values in the enttyp1 and entypds nodes of the schema. The metadata elements within the attributes have a name and source based on the values in the attrlabl and attrdefs nodes of the attr nodes within the detailed node. All of the metadata attribute and element definitions can be found in the lead_attribute_definition and lead_element_definition tables in the myLEAD database.

Following is a mapping from tags in the LMS to metadata attribute and element names in myLEAD. The indented bullets are the metadata elements. This list excludes the dynamic metadata attributes and elements.
<table>
<thead>
<tr>
<th>LEAD Schema Element Tag</th>
<th>myLEAD Attribute Name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Element Name</strong></td>
</tr>
<tr>
<td>citation</td>
<td>citation</td>
</tr>
<tr>
<td>• origin</td>
<td>• originator</td>
</tr>
<tr>
<td>• pubdate</td>
<td>• pubDate</td>
</tr>
<tr>
<td>• pubtime</td>
<td>• pubTime</td>
</tr>
<tr>
<td>• title</td>
<td>• title</td>
</tr>
<tr>
<td>• edition</td>
<td>• edition</td>
</tr>
<tr>
<td>• geoform</td>
<td>• geoFormat</td>
</tr>
<tr>
<td>• sername</td>
<td>• seriesName</td>
</tr>
<tr>
<td>• issue</td>
<td>• seriesIssue</td>
</tr>
<tr>
<td>• pubplace</td>
<td>• pubPlace</td>
</tr>
<tr>
<td>• publish</td>
<td>• publisher</td>
</tr>
<tr>
<td>• othercit</td>
<td>• otherCite</td>
</tr>
<tr>
<td>• onlink</td>
<td>• onlink</td>
</tr>
<tr>
<td>• descript</td>
<td>• description</td>
</tr>
<tr>
<td>• abstract</td>
<td>• abstract</td>
</tr>
<tr>
<td>• purpose</td>
<td>• purpose</td>
</tr>
<tr>
<td>• supplinf</td>
<td>• supplementalInfo</td>
</tr>
<tr>
<td>• status</td>
<td>• status</td>
</tr>
<tr>
<td>• progress</td>
<td>• progress</td>
</tr>
<tr>
<td>• update</td>
<td>• update</td>
</tr>
<tr>
<td>• accconst</td>
<td>• accessConstraint</td>
</tr>
<tr>
<td>• useconst</td>
<td>• useConstraint</td>
</tr>
<tr>
<td>• keywordTheme</td>
<td>• keywordTheme</td>
</tr>
<tr>
<td>• themekt</td>
<td>• themeThesaurus</td>
</tr>
<tr>
<td>• themekey</td>
<td>• themeKeyword</td>
</tr>
<tr>
<td>• placekt</td>
<td>• placeThesaurus</td>
</tr>
<tr>
<td>• placekey</td>
<td>• placeKeyword</td>
</tr>
<tr>
<td>• stratum</td>
<td>• stratumThesaurus</td>
</tr>
<tr>
<td>• stratkt</td>
<td>• stratumKeyword</td>
</tr>
<tr>
<td>• stratkey</td>
<td></td>
</tr>
</tbody>
</table>
## LEAD Schema Element Tag

<table>
<thead>
<tr>
<th>LEAD Schema Element Tag</th>
<th>myLEAD Attribute Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>temporal</td>
<td>keywordTemporal</td>
</tr>
<tr>
<td>tempkt</td>
<td>temporalThesaurus</td>
</tr>
<tr>
<td>tempkey</td>
<td>temporalKeyword</td>
</tr>
<tr>
<td>metainfo</td>
<td>metadataInfo</td>
</tr>
<tr>
<td>metd</td>
<td>metadataDate</td>
</tr>
<tr>
<td>metstdn</td>
<td>metadataStdName</td>
</tr>
<tr>
<td>metstdv</td>
<td>metadataStdVersion</td>
</tr>
<tr>
<td>timeperd</td>
<td>timePeriod</td>
</tr>
<tr>
<td>current</td>
<td>timePeriodCurrentness</td>
</tr>
<tr>
<td>caldate (sngdate)</td>
<td>calendarDate (sngdate)</td>
</tr>
<tr>
<td>time (sngdate)</td>
<td>calendarTime (sngdate)</td>
</tr>
<tr>
<td>begdate (rngdates)</td>
<td>beginDate (rngdates)</td>
</tr>
<tr>
<td>begtime (rngdates)</td>
<td>endTime (rngdates)</td>
</tr>
<tr>
<td>enddate (rngdates)</td>
<td>dateBegin (rngdatesInd)</td>
</tr>
<tr>
<td>endtime (rngdates)</td>
<td>dateEnd (rngdatesInd)</td>
</tr>
<tr>
<td>begin (rngdatesInd)</td>
<td>duration (rngdatesInd)</td>
</tr>
<tr>
<td>end (rngdatesInd)</td>
<td></td>
</tr>
<tr>
<td>duration (rngdatesInd)</td>
<td></td>
</tr>
<tr>
<td>bounding</td>
<td>spatialBounds</td>
</tr>
<tr>
<td>westbc</td>
<td>westCoordinate</td>
</tr>
<tr>
<td>eastbc</td>
<td>eastCoordinate</td>
</tr>
<tr>
<td>northbc</td>
<td>northCoordinate</td>
</tr>
<tr>
<td>southbc</td>
<td>southCoordinate</td>
</tr>
<tr>
<td>vertdom</td>
<td>spatialVertical</td>
</tr>
<tr>
<td>lowerb</td>
<td>lowerBound</td>
</tr>
<tr>
<td>upperb</td>
<td>upperBound</td>
</tr>
<tr>
<td>overview</td>
<td>eaOverview</td>
</tr>
<tr>
<td>eaover</td>
<td>eaOverview</td>
</tr>
<tr>
<td>eadetcite</td>
<td>eaDetailCite</td>
</tr>
</tbody>
</table>

* The boundingBox metadata element defined in myLEAD is a spatial element based on the four coordinates in the spatialBounds metadata attribute. This allows for spatial queries over the bounding coordinates (e.g., “does boundingBox contain the lat/lon point 35,-90?”).
<table>
<thead>
<tr>
<th><strong>LEAD Schema Element Tag</strong></th>
<th><strong>myLEAD Attribute Name</strong></th>
<th><strong>Element Name</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>cntinfo</td>
<td>distContact</td>
<td>• contactPerson</td>
</tr>
<tr>
<td>• cntper</td>
<td></td>
<td>• contactOrg</td>
</tr>
<tr>
<td>• cntorg</td>
<td></td>
<td>• contactPos</td>
</tr>
<tr>
<td>• cntpos</td>
<td></td>
<td>• contactPhone</td>
</tr>
<tr>
<td>• cntvoice</td>
<td></td>
<td>• contactTTD</td>
</tr>
<tr>
<td>• cnttdd</td>
<td></td>
<td>• contactFax</td>
</tr>
<tr>
<td>• cntfax</td>
<td></td>
<td>• contactEMail</td>
</tr>
<tr>
<td>• cntemail</td>
<td></td>
<td>• contactHours</td>
</tr>
<tr>
<td>• hours</td>
<td></td>
<td>• contactInstruction</td>
</tr>
<tr>
<td>• cntInst</td>
<td></td>
<td>• cntAddrType</td>
</tr>
<tr>
<td>• addrtype (cntaddr)</td>
<td></td>
<td>• cntAddrLine</td>
</tr>
<tr>
<td>• address (cntaddr)</td>
<td></td>
<td>• cntAddrCity</td>
</tr>
<tr>
<td>• city (cntaddr)</td>
<td></td>
<td>• cntAddrState</td>
</tr>
<tr>
<td>• state (cntaddr)</td>
<td></td>
<td>• cntAddrZip</td>
</tr>
<tr>
<td>• postal (cntaddr)</td>
<td></td>
<td>• cntAddrCountry</td>
</tr>
<tr>
<td>• country (cntaddr)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>stdorder</td>
<td>stdOrderProcess</td>
<td>• formatName</td>
</tr>
<tr>
<td>• formname</td>
<td></td>
<td>• formatVersionNum</td>
</tr>
<tr>
<td>• formvern</td>
<td></td>
<td>• formatVersionDate</td>
</tr>
<tr>
<td>• formverd</td>
<td></td>
<td>• formatSpec</td>
</tr>
<tr>
<td>• formspec</td>
<td></td>
<td>• formatInfoContent</td>
</tr>
<tr>
<td>• formcont</td>
<td></td>
<td>• fileDecompression</td>
</tr>
<tr>
<td>• filedec</td>
<td></td>
<td>• transferSize</td>
</tr>
<tr>
<td>• transize</td>
<td></td>
<td></td>
</tr>
<tr>
<td>complete</td>
<td>dataQualComplete</td>
<td>• dataQualComplete</td>
</tr>
<tr>
<td>procsstep</td>
<td>dqProcessStep</td>
<td>• dqProcessDesc</td>
</tr>
<tr>
<td>• procdesc</td>
<td></td>
<td>• dqResourceID</td>
</tr>
<tr>
<td>• resourceId</td>
<td></td>
<td>• dqProcessDate</td>
</tr>
<tr>
<td>• procDateTime</td>
<td></td>
<td>• dqResourceID</td>
</tr>
<tr>
<td>• resourceId (srcused)</td>
<td></td>
<td>• dqSourceType</td>
</tr>
<tr>
<td>• srctype (srcused)</td>
<td></td>
<td>• dqResourceID</td>
</tr>
<tr>
<td>• resourceId (srcprod)</td>
<td></td>
<td>• dqSourceType</td>
</tr>
<tr>
<td>• srctype (srcprod)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• resourceId (enclosedresources)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>resourceID (enclosedresources)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Appendix B – LEAD Metadata Schema (LMS)

On the following pages is a diagram of the LMS. As noted in the color-key on the first page of the diagram, the blue nodes that are not italicized are metadata attributes. The italicized blue nodes are sub-attributes.
LEAD Schema
Color Coded Shredding Into myLEAD

Symbol Key

Optional element, but if used, it can have required child elements.

Optional element that can have multiple instances.

Required element, but only one instance.

Required element, but multiple instances allowed.

Sequence. When an element has children, they are contained within a sequence. A document must keep children in the same order shown in the sequence.

Optional Sequence. Can have both optional and required elements, so if the optional sequence is included, all required child elements must be included. The symbol shown here allows multiple instances. A required sequence could also allow for multiple instances.

Choice. Only one of the children of a choice can be included.

Optional Choice. One or none of the children can be included.

NOTE: A Choice can have multiple instances - in that case the order of the children in the document can vary.

Metadata Color Key

Structural Metadata Attribute

Structural Metadata Sub-Attribute

Structural Metadata Element

Structural Metadata Attribute and Element

Dynamic Metadata Attribute Name

Dynamic Metadata Attribute Source

Dynamic Metadata Sub-Attribute

Dynamic Metadata Element/Sub-Attribute Name

Dynamic Metadata Element/Sub-Attribute Source

Dynamic Metadata Element Value

1 These attributes are also stored as CLOBs in myLEAD