



LINKED ENVIRONMENTS FOR ATMOSPHERIC DISCOVERY

MyLead Release V0.3 alpha myLEAD Client Developer's Guide

Project Title: myLEAD

Document Title: myLEAD Release V0.3 alpha Installation Guide

Organization: Indiana University

Date: May, 27, 2005

Contact: Beth Plale (plale@cs.indiana.edu)

Authorship: Beth Plale, Sangmi Lee Pallickara, Scott Jensen, and Yiming Sun

1	Introduction.....	2
1.1	myLEAD License	2
2	Setting Up a myLEAD User	3
3	Using the myLEAD Client.....	3
3.1	Accessing the myLEAD Client.....	3
3.2	Working With The Lead Client	4
3.2.1	Adding Metadata to myLEAD Using the myLEAD Client.....	4
3.2.1.1	Using Dynamic Attributes	5
3.2.1.2	Adding Metadata Example	6
3.2.1.3	Adding Attributes.....	8
3.2.2	Querying the myLEAD Catalog	8
3.2.3	Using the QueryLead Method.....	8
3.2.4	Administrative Functions.....	10

1 Introduction

This document provides an overview of using the myLEAD Client to add metadata to the myLEAD catalog or query existing data in the 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 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 `mcs_lead` database underlying the myLEAD catalog. There are client administrative methods that can be used to add subsequent users. The administrative interface cannot be used for adding the first user because as each user is added, the system records who added the user and when, so there has to be a first user to add additional users.

To add the user in myLEAD, enter MySQL and then type the following at the `mysql>` prompt to use the `mcs_lead` database:

```
use mcs_lead;
```

The table containing user information is named `mcs_writer`. While this table has a number of fields that are included to hold data needed for a document to conform to the FGDC-based LEAD schema, the `Writer_Dn` field is the only field in the table that must be populated. The `dn` must be unique. To add the first user to the catalog, use the following insert query:

```
INSERT INTO mcs_writer (Writer_Dn) VALUES ("my new dn");
```

Where the string `my new dn` is replaced by the `dn` you wish to use.

3 Using the myLEAD Client

The myLEAD client will generally be accessed through the myLEAD Agent, but this section of the guide provides information on accessing the myLEAD client directly and the Java API that can be used with the myLEAD client.

3.1 Accessing the myLEAD Client

The myLEAD client and related API are Java-based, so you will need to import the following two myLEAD packages into your Java code:

```
edu.indiana.dde.mylead.client.*  
edu.indiana.dde.mylead.client.*
```

Both of these packages are included in the `leaddai.jar` file which is installed as part of the myLEAD Client installation. Since myLEAD extends MCS, you will also need to import the following jar files related to MCS:

```
org.globus.mcs.client.MCSException  
org.globus.mcs.common.ReturnType
```

In your program code, you can create an instance of `LeadClient`. There are two version of the of the `LeadClient` constructor, both of which require the path to the registry as the third parameter. Following is an example of the path to the registry that assumes the server is on localhost and port 8080. You will need to modify this for the location of your registry and the port you set when installing OGSA-DAI and the myLEAD software.

```
http://localhost:8080/ogsa/services/ogsadai/DAIServiceGroupRegistry
```

The first two parameters for the constructor can be null as long as the registry is passed as a parameter. If you are satisfied with the default life, which is currently set to 5 minutes, then you can create a new `LeadClient` with the following code:

```
String registry = "http://localhost:8080/ogsa/services/ogsadai/DAIServiceGroupRegistry";  
LeadClient client = new LeadClient(null,null,registry);
```

If you want to have a different lifespan, you can set the number of minutes for the life span as the fourth parameter to the constructor.

3.2 Working With The Lead Client

The `leadClient` has a number of methods that can be used to create new metadata objects in myLEAD and query for previously added metadata. Many of these methods are overloaded to allow for alternate approaches and most take objects from the myLEAD client API as parameters. For purposes of the following discussion, the methods are divided into three categories:

- Adding Metadata
- Querying the Catalog
- Administrative Functions

The interface for the myLEAD Client is defined in the `LeadClientIntf` class in the `edu.indiana.dde.mylead.client` package.

3.2.1 Adding Metadata to myLEAD Using the myLEAD Client

The methods in the myLEAD Client for adding metadata are all named `create` but take different parameters from the Java API that determine the type of data being added:

```
create(String dn, MyLeadData item)  
create(String dn, MyLeadData item, String parentName, String parentType)
```

In addition, to add attributes to an existing metadata object (such as a file or collection definition) the following method is used:

```
addAttribute(String dn, MyLeadAttrData attribute, String parentName,  
String parentType)
```

All of the above methods return an instance of `ReturnType` which is defined in MCS and indicates whether the metadata was successfully added. Each of these also throws an `MCSException` that must be caught or thrown by any program using these client methods.

In the `create` method, the `MyLeadData` parameter is an object from the myLEAD client API that can contain metadata regarding a file, collection, experiment, or project. `MyLeadData` is an abstract base class which is defined in the `edu.indiana.dde.mylead.common` package. To define the metadata for a file, collection, experiment, or project, create an instance of one of the following classes which inherit from the `MyLeadData` base class:

```
MyLeadFileData
MyLeadCollData
MyLeadExpData
MyLeadProjData
```

Each of these classes contains the following two methods which provide the ability to define some of the static attributes in the metadata for their respective types:

```
setName(String nameStr)
setDesc(String descStr)
```

In addition, all of the classes except `MyLeadFileData` contain the following additional method that provides the ability to set a logical data related to a collection, experiment, or project.

```
setDate(Date myDate)
```

When metadata is initially created for any of these four types, the system will also automatically record the user that added the metadata definition as well as the date and time that it was inserted.

In addition to the static attributes, currently most of the attributes in myLEAD are dynamic attributes. The difference is that while adding or modifying the definition of a static attribute would require a database change, additional dynamic attributes can be defined with a simple insert and then used to add metadata regarding any file, collection, experiment, or project.

3.2.1.1 Using Dynamic Attributes

Each of the four classes listed above has an additional method for adding dynamic attributes:

```
addAttribute(MyLeadAttrData leadAttribute)
```

The single parameter, `(MyLeadAttrData)` is also defined in the `edu.indiana.dde.mylead.common` package.

The class `MyLeadAttrData` contains the following methods:

```
setName(String attrName)
addElement(MyLeadElemData leadElement)
addGeoElement(MyLeadGeoData leadGeoElement)
addAttribute(MyLeadAttrData leadAttribute)
```

As can be seen from the above list of methods, attributes can contain subattributes. In addition, each attribute can contain any number of elements. Each element is a single data value that is one of eight types: string, integer, float, date, datetime, time, spatial, or text (blob). When an attribute is defined, the elements and their data types are also defined. Also, for attributes and elements, a name value is defined which is the text used for the tag when displaying metadata in an XML format as well as a description and short description.

The definitions for the attributes already defined in myLEAD are in the `lead_attribute_definition` and `lead_element_definition` tables

respectively. The `leadClient` also has a method named `attributeDef` which will return an XML document that shows the structure of all of the elements and attributes currently defined in myLEAD.

3.2.1.2 Adding Metadata Example

Following is a brief example which shows the steps required if we wanted to add a new project to the myLEAD catalog which contained an experiment, which in turn contained a collection with a file. In addition, the project contains a “topic” attribute, (similar to theme in FGDC), the collection has attributes for use constraints, and the file has attributes for mean sea level pressure, and dry bulb temperature. The attributes defined in myLEAD currently are prior to development of the LEAD schema. The attribute definitions will be updated to the LEAD schema in a subsequent version of myLEAD.

First we need to create a new project definition:

```
MyLeadProjData testProject = new MyLeadProjData();
```

We then can set the static attributes for the project. In this case we will set the name, description, and GUID. We will then also add a simple topic attribute that indicates this project relates to mesoscale weather. The Topic is a simple string attribute, and the name of both the attribute and element definition is “Topic”. The code needed to define the project is:

```
testProject.setName("the name of my project");
testProject.setDesc("A longer description of my project");
testProject.setGUID("LEADID:lead.org:bc6e900-cd8cb84-11d9-8cd5-0800200c9a66:1");
//Create a new attribute and add an element to it
MyLeadAttrData testProjAttr = new MyLeadAttrData("Topic");
testProjAttr.addElement(new MyLeadElemData("Topic", "Mesoscale Weather"));
//Add the attribute to the project definition
testProject.addAttribute(testProjAttr);
```

In the `MyLeadProjData` class, there are methods that can be called to add an experiment, collection, or file as a member of the project. Likewise, the experiment and collection classes have methods to add a collection or file (note that collections can be nested arbitrarily deep).

To define the metadata for our file, collection, and experiment we would write the following code:

```
//add metadata for experiment
MyLeadExpData testExp = new MyLeadExpData();
testExp.setName("My weather experiment");
testExp.setDesc("My first mesoscale weather forecast");
testExp.setGUID("LEADID:lead.org:bc6e900-cd4ae71-14b-8a16-0832240dda66:1");

//add metadata for collection
MyLeadCollData testCol = new MyLeadCollData();
```

```
testCol.setName("My collection of data files");
testCol.setDesc("data files for my first mesoscale weather project");
testCol.setGUID("LEADID:lead.org:bea34c0-cdaa501-1cb-8a16-
0832240dda66:1");

//add attribute to collection for accessconstraints
MyLeadAttrData testColAttr = new MyLeadAttrData("AccessCon");
testColAttr.addElement(new MyLeadElemData("AccessCon",
"Noncommercial"));
testCol.addAttribute(testColAttr);

//add file with attributes
MyLeadFileData testFile = new MyLeadFileData();
testFile.setName("My data file");
testFile.setDesc("My data file with humidity and temperature data");
testFile.setGUID("LEADID:lead.org:bc6fc21-ceb4502-9a5-8a16-
0832240dda66:1");

//add file attribute for dry bulb temp
MyLeadAttrData attr1 = new MyLeadAttrData("DBT");
attr1.addElement(new MyLeadElemData("DbtTemp", 40.5));
attr1.addElement(new MyLeadElemData("DbtUom", "Celsius"));
testFile.addAttribute(attr1);

//add file attribute for mean sea level pressure
MyLeadAttrData attr2 = new MyLeadAttrData("MSLP");
attr2.addElement(new MyLeadElemData("MSLP", 1.2));
testFile.addAttribute(attr2);

//add file to collection
testCol.addFile(testFile);

//add collection to experiment
testExp.addCollection(testCol);

//add the experiment to the project
testProject.addExperiment(testExp);
```

After the above code, we have all of the metadata for a project that contains an experiment, which contains a collection, that contains a file. To add all of this to the myLEAD catalog, we would call the client instance's create method:

```
client.create("my dn", testProject);
```

In the above example, we used the version of create that takes only two parameters since the parameter we are passing is a project and it has no parent object that contains it. If we were instead adding a collection of derived files that were the result of a calculation to an existing experiment definition, we would instead have used the versions of create that takes four parameters - the two additional parameters being the name of the experiment and the string "EXPERIMENT" which indicates that the parent is an experiment. There are four types and they are defined in the leadActivity.xsd schema that is used to validate the perform documents for inserting metadata.

When we call the create method, behind the scene myLEAD will call methods on the classes from the myLEAD API that will wrap up the metadata into a perform document that is then sent by myLEAD to the server. The return type will indicate if the data was added to the catalog successfully.

3.2.1.3 Adding Attributes

Similar to adding a collection to an existing experiment, the `addAttribute` method in the myLEAD client takes four parameters, the same four parameters as the `create` method except that instead of passing an instance of `MyLeadCollData` as the second attribute, an instance of `MyLeadAttrData` is passed that contains the metadata for a new attribute - populated in the same manner as done above when creating a new project.

3.2.2 Querying the myLEAD Catalog

The main method for querying in myLEAD is the `queryLead` method. In addition to this method, there are two other methods:

```
userExists(String dn)
attributeDef(String dn, LeadStringHolder attrDef)
```

The `userExists` method is probably self-explanatory. When called with a `dn` parameter, it will return a `ReturnType` of either `OPERATION_SUCCESSFUL` which indicates that the user's profile exists in the `mcs_writer` table or `NO_RESULTS_FOUND` which means the user's profile could not be found.

The other method, `attributeDef` was mentioned above. This method will return an XML fragment that contains all of the attributes and elements currently defined within the database. The elements will be nested within the attributes that they are defined for. As mentioned earlier, an attribute can also have subattributes. In that case, the definitions for the subattributes are also nested within the attributes they are defined for. The `LeadStringHolder` parameter for this method is a container that is used to return the string from the server.

3.2.3 Using the QueryLead Method

The main myLEAD method to query for metadata is the `queryLead` method, and it has two possible forms:

```
queryLead(String dn, int limit, String filter, MyLeadQuery target,
LeadStringHolder results)
```

```
queryLead(String dn, int limit, String filter, String target, String
query, LeadStringHolder results)
```

In both forms, the first three parameters are the same. As in adding metadata to the catalog, the `dn` parameter is the `dn` of the user adding the data. When users add data to the catalog, it is tagged with their `dn`, and only they can query for it again. The metadata added by each user remains private to that user. In the `mcs_writer` table, an internal unique ID is assigned to each user, so if their `dn` should change at a future date, the administrator could update their record in the `mcs_writer` table and all of

their previously entered metadata would automatically be associated with the revised dn.

The second parameter in the query is the `limit`. This allows the user to limit the number of records returned which can reduce response time and reduce the burden to the server. This could be beneficial if the user is looking for an input template or file that may be associated with many of their experiments and they just needed to find a copy of it – they could then set the limit to 1.

The third parameter in both `queryLead` methods is the `filter`, but we will defer discussing the `filter` until after discussing the target.

Both versions of this method have a `target` parameter, but one is an instance of `MyLeadQuery` and the other is a `String`. The class `MyLeadQuery` is defined as an abstract base class in the `edu.indiana.dde.mylead.common` package and provides a means of specifying the type of object to query for and parameters about it. For the four different object types, there are the following classes that inherit from `MyLeadQuery`:

```
MyLeadFile  
MyLeadCollection  
MyLeadExperiment  
MyLeadProject
```

Each of these classes has the following methods to allow the user to query based on static and dynamic attributes:

```
addName(String name, boolean exact)  
addDesc(String desc, boolean exact)  
creatorSearchRng(Date startDate, Date endDate)  
setStartCreateRng(Date startDate)  
setEndCreateRng(Date endDate)  
addAttribute(MyLeadAttribute leadAttribute)
```

The “add” methods allow a user to specify one or more possible object names or descriptions for possible matches. If the `exact` parameter is set to `true`, then the name or description needs to be an exact match, otherwise it can be a fragment of the name or description. The creator search methods allow the user to specify a range within which the create date for the object should be. If only a start date is specified, then the end date is unbounded (and likewise if only an end date is specified then the start date is unbounded).

The `addAttribute` method allows the user to specify attributes that the object should have. Any attributes specified provide a minimum condition for an object to match the query. The `MyLeadAttribute` class has methods to specify element conditions on the attributes. If an attribute is defined to have 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 them. For the elements, there are constructor methods that allow for the following possibilities:

Match a string element – partial match is based on the `exact` parameter:

```
MyLeadElement(String name, String attrValue, boolean exact)
```

Comparison to a value:

```
MyLeadElement(String name, String attrValue, int compare)
```

In this method the compare parameter is one of the following constants defined in LeadConstants:

```
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
```

Within a range:

```
MyLeadElement(String name, String attrStart, String attrEnd)
```

The start and end attributes are strings in the method, but they should be representative of the data type for the specific element. In performing the comparison, myLEAD first tries to convert the string to the relevant data type (e.g., date, time, integer, float). If the element cannot be converted, then the element is discarded as a criteria for the query.

Based on the example used for adding metadata to the catalog, the following code would create a target that looks for a file that has the attribute "dry bulb temperature" with a value greater than 35 degrees Celsius:

```
MyLeadFile qryFile = new MyLeadFile();
MyLeadAttribute dbtAttr = new MyLeadAttribute("DBT");
dbtAttr.addElement(
new MyLeadElement("DbtTemp", "35", LeadConstants.
MYLEAD_GREATER_THAN));
qryFile.addAttribute(dbtAttr);
```

If the leadQuery were executed with this target, it would find the file we added earlier as part of our project. When the query is executed, the MyLeadFile class will call methods that wrap up all of the parameters for the query in a perform document matching the leadActivity.xsd schema. The other version of the leadQuery method allows for the user to create the XML fragment on their own instead of using the API.

3.2.4 Administrative Functions

The administrative functions available in the client provide the ability to add / edit user profiles and add / edit information regarding the storage preferences and replica location that a user is assigned.

Consistent with the methods for adding metadata to the catalog, there methods for defining a new user or replica are also named create but take different classes from the client API as parameters:

```
create(String dn, MyLeadReplica replica)
create(String dn, MyLeadUser user)
```

There are also methods for updating an existing user profile or replica definition:

```
update(String dn, MyLeadReplica replica)
    update(String dn, MyLeadUser user)
```

To query for user profiles or replicas defined in myLEAD, the following query methods are provided:

```
queryReplica(String dn, LeadStringHolder results)  
queryUser(String dn, LeadStringHolder results)
```

All of the administrative methods return a `ReturnType` which indicates either success or failure. All of the methods also can throw a `MCSEException` that needs to be handled or throw.