



MyLead Release V0.3 alpha Developer's Guide

Project Title: myLEAD

Document Title: myLEAD Release V0.3 alpha Developer's Guide

Organization: Indiana University

Date: June, 13, 2005

Contact: Sangmi Lee Pallicaka (leesangm@cs.indiana.edu)

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

1	Introduction.....	2
1.1	myLEAD License	2
2	System Specification.....	3
2.1	Prerequisite Products	3
2.2	Required Libraries	3
3	Access to the MyLead Agent Service.....	4
3.1	Generating and Parsing Message	4
4	MyLead Agent APIs	4
4.1	Active Experiment APIs	4
4.1.1	Creating and Closing Experiment.....	5
4.1.2	Setup the experiment.....	6
4.1.3	Create collection	6
4.1.4	Register a File	6
4.1.5	Query APIs about Active Experiment	7
4.2	Query APIs (Non-session requests).....	7
4.3	Administrative APIs.....	8

1 Introduction

This document provides a guide to the developers accessing mylead server through the mylead agent service. The document covers:

- How to setup the client environment
- How to access the mylead agent service
- Description of mylead agent APIs

You can start with downloading `myleadStarterKit0.3alpha.tar.gz` from distribution site. This sample code is also available in mylead agent service package.

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 System Specification

myLEAD is designed to work on - and has been tested under following platforms:

- Redhat Linux Gentoo

2.1 Prerequisite Products

Before you start develop a requester to mylead agent service, the mylead system should be installed and running properly. The mylead system consists,

- JDK 1.4X
- ANT
- Globus Replica Location Service (RLS)

Please see <http://www.isi.edu/~annc/rls/INSTALL.html>

- WS Notification/Eventing Broker

WS-Messenger(WSMG) is WS-Notification & WS-Eventing implementation.

See <http://www.extreme.indiana.edu/xgws/messenger/index.html>

About the installation of the mylead system, please refer the installation guide.

2.2 Required Libraries

JAR Name	Used For	Package Version	URL
xmlbeans-1.0.jar	Process XML message	1.0	http://xmlbeans.apache.org/
xpp3-1.1.3.4.N.jar	XML parser	3-1.1.3.4.N	http://www.extreme.indiana.edu/xgws/xsoap/xpp/mxp1/index.html
Myleadagent-0.3alpha-interface.jar	MyLead agent interface	0.3 alpha	In mylead distribution site
GSX-1.2_SC04_20041102.jar	Grid toolkit	1.2	http://www.extreme.indiana.edu/xgws/GSX/

3 Access to the MyLead Agent Service

First, please make sure you have valid account in mylead server. Also make sure that mylead server, agent service, RLS server and notification(WS-Eventing brokering service) server are accessible. To test your setup, mylead installation test suite is available in the distribution site. Please download myleadInstallationTest0.3alpha.tar and check your environment. Installation test suite is also available in the agent service package. For more information, please refer the installation guide.

To access mylead agent service, service requesters should include,

```
import edu.indiana.extreme.gsx.util.ServiceInvoker;
import edu.indiana.extreme.lead.types.*;
import edu.indiana.dde.mylead.agent.MyLeadAgentInterface;
```

The mylead agent service is invoked by ServiceInvoker. For example,

```
MyLeadAgentInterface myleadAgentService =
    (MyLeadAgentInterface)ServiceInvoker.
    createXBeansRpcProxy(service_url,MyLeadAgentInterface.class);
```

3.1 Generating and Parsing Message

MyLead agent service follows data schema used in All-hands-meeting in May, 12, 2005. Each XML schema is generated and parsed by Java APIs generated by XMLBeans. Class files, source files, and schema are available at download page. Please note that this schema may change after the LEAD schema is published.

MyLead client package includes sample code, `ClientTestMessageGenerator.java`.

4 MyLead Agent APIs

The myLEAD v 0.3 alpha provides,

- Active Experiment APIs
- Basic Query APIs
- Administrative APIs

4.1 Active Experiment APIs

MyLead agent service maintains the finite state machine to monitor and manage activities related to creating new collections and registering files. A set of APIs processed by FSM is named as “*Active Experiment APIs*”. Requests associated to the active experiment are queued in the *request queue* in mylead agent service.

The requests are processed in the sequence of,

Creating an active experiment

- Setup the active experiment
- Create collection and register files
- Workflow finished or close/shutdown the active experiment

Above sequence is not exactly same with the finite state machine in the mylead agent service. They are simplified steps for developers. For more information about the finite state machine, please refer our publication¹.

4.1.1 Creating and Closing Experiment

```
public String createExperiment(String ulD, String proj, MyLeadExperimentDocument exp)
```

The active experiment is started by `createExperiment()` method. Requesters should decide project name and experiment name before calling this method. `createExperiment()` requests three arguments, user name, project name, and experiment name. This tuple will identify new experiment. For mylead v0.3alpha, each project and experiment should have unique name under a user. The user name must be a valid mylead account. In general, it is user's DN.

```
public void shutdownExperiment(String ulD, String proj, String exp)
```

There are three ways to finish active experiment. If the mylead agent receive `COMPLETE_WORKFLOW` or `INCOMPLETE_WORKFLOW` notification from workflow engine, the active experiment will be closed automatically. Please note that `COMPLETE_WORKFLOW/INCOMPLETE_WORKFLOW` is about the activity of the whole workflow activity which is not related to the activities of workflow components. MyLead agent requires notification from workflow engine, when

- the workflow engine starts its workflow tasks (`START_WORKFLOW`)
- the workflow engine finishes its workflow tasks (`COMPLETE_WORKFLOW/INCOMPLETE_WORKFLOW`)

Besides the notification from workflow engine, requesters can close their experiment. `shutdownExperiment()` closes an active experiment immediately. If there are some requests in the request queue, they will throw exceptions.

```
public void closeExperiment(String ulD, String proj, String exp)
```

Meanwhile, if the requester wants to close the active experiment, but still wants to process all of the requests in the request queue, the requester can call this method.

¹ Sangmi Lee Pallickara, Beth Plale, Scott Jensen, Yiming Sun, Monitoring Access to Stateful Resources in Grid Environments, *To appear IEEE International Conference on Services Computing*, Orlando, Florida, July 2005

4.1.2 Setup the experiment

```
public void setWorkflowTemplate(String uID, String proj, String exp,LeadEntryDocument workflowTemplate)
```

Each experiment allows setting attributes about the experiment. `setWorkflowTemplate()` sets a workflow template that will be used for this active experiment. Requesters must generate `LeadEntryDocument` properly. Please refer the sample code in `ClientTestMessageGenerator.java`.

```
public void setWorkflowInstance(String uID, String proj, String exp,LeadEntryDocument workflowInstance)
```

This method sets the workflow instance.

4.1.3 Create collection

```
public void createCollection(String uID, String proj, String exp, String parent,String collectionName, String collectionDesc)
```

The requesters are allowed to create their own collections. MyLead also supports nested collections. For example, if you want to create `collectionB` under `collectionA`, create `collectionB` with the argument `parent` as `collectionA`. If `parent` is null, `createCollection()` will generate an empty collection under the experiment.

4.1.4 Register a File

```
public void registerFile(String uID, String proj, String exp, String collectionName,String fileName, String fileDesc, edu.indiana.extreme.lead.types.LeadEntryDocument dataset)
```

MyLead 0.3 alpha requires the requesters register their files in RLS first. Therefore, if the mylead agent cannot find the file (query by the `fileName`), it will throw exception. MyLead 0.3 alpha provides two options to store files under collection. You can store file under the collection you defined, or collection automatically generated for the input files. To register a file to mylead under the collection you defined earlier, you should pass the collection name as the argument `collectionName`. If the `collectionName` is null, mylead will register the file under the default input data file collection, “[experiment name]:input data” For this version, metadata is flowing the schema defined for SC04 demo, this is a subject to change after the LEAD schema is fixed.

4.1.5 Query APIs about Active Experiment

MyLead provides some useful query APIs about active experiments. This information is available only the active experiment is in the process. After the requester close or shutdown the experiment or the workflow engine finishes its task, this information is not available any more. MyLead throws exceptions.

```
public String isActiveExperiment(String uid, String proj, String exp)
public String getActiveWorkflowTemplateName(String uid, String proj, String exp)
public String getActiveWorkflowInstanceName(String uid, String proj, String exp)
public String getCurrentState(String uid, String proj, String exp)
public String getActiveNotificationTopicID(String uid, String proj, String exp)
```

4.2 Query APIs (Non-session requests)

Query APIs do not require creating an active experiment. Whether the experiment is still in active experiment, or already finished, requesters can access the information stored in mylead server through these APIs.

```
public void createProject(String uid, String proj, String projdesc)
```

This method creates a new project under the user ID. If there is no mylead account under the user id or there is a project with same name already, this will throw an exception.

```
public MyLeadInvestigationArrayDocument getProjects(String uid)
```

This method returns all project under the userid. To parse the return value, MyLeadInveestigationArrayDocument, please utilize data schema jar file at <http://www.cs.indiana.edu/~leesangm/mylead>.

```
public void addWorkflowTemplate(String uid, String proj, LeadEntryDocument workflowTemplate)
```

This method creates a new workflow template under the project. Requesters should build a LeadEntryDocument contains information about the workflow template. Please refer the sample file, ClientTestMessageGenerator.java

```
public LeadEntryArrayDocument getWorkflowTemplates(String uid, String proj, String wftName, String wftDesc)
```

This method returns all workflow template stored under the user id.

```
public StringArrayDocument getWorkflowTemplateName(String uid, String proj)
```

This method returns only names of the workflow template stored under the user id.

```
public LeadEntryArrayDocument getWorkflowInstance(String uid, String inv, String expName, String workflowName)
```

This method returns workflow instances which have name matching with the workflowName. The return value will include all workflow instances under the userid containing the workflowName as part of their names.

```
public MyLeadExperimentArrayDocument getExperiments(String uid, String proj, int lastNDays)
```

This method returns all experiments under the project created in NDays.

```
public WorkflowNotificationArrayDocument getNotifications(String uid, String proj, String exp, int countFromLast)
```

This method returns notifications stored under the experiment. The return value will include only countFromLast number of notifications from the latest notification.

```
public String getWorkspace(String uid)
```

This method returns user's workspace. The return value is OGSA-DAI perform document as a String. MyLead agent service modified original perform document for myworkspace portlet.

4.3 Administrative APIs

MyLead v0.3 alpha provides a set of APIs for mylead administrator.

```
public MyLeadServiceInfoArrayDocument getMyLeadServiceInfo()
```

This method provides the information about mylead distributed services. MyLead system will be deployed as a master-satellite like distributed service in the future release. This API provides information about all MyLead systems over the MyLead network.

```
public void setMyLeadServiceInfo(MyLeadServiceInfoDocument myleadsrvinfo)
```

This method modifies the information of mylead system which already exists.

```
public void addMyLeadServiceInfo(MyLeadServiceInfoDocument myleadsrvinfo)
```

This method creates a mylead system which is newly joined in the distributed mylead system.

```
public MyLeadUserInfoArrayDocument getAllUserInfo()
```

This method returns user information of all of distribute mylead system.

```
public MyLeadUserInfoDocument getUserInfo(String userid)
```

This method queries information about single user with `userid`.

```
public void setUserInfo(MyLeadUserInfoDocument userinfo)
```

This method modifies a user's information which already exists in the mylead system.

```
public void addUserInfo(MyLeadUserInfoDocument userinfo)
```

This method creates a new user in the mylead system.

```
public StringArrayDocument getActiveExperiment(String uID, String proj)
```

This method returns all of the active experiments running in the mylead system.

```
public StringArrayDocument getActiveProject(String uID)
```

This method returns all of the active projects running in the mylead system.

5 Sample Code

`ClientTest.java` : `src/java/sample/ClientTest.java`

```
import edu.indiana.extreme.gsx.util.ServiceInvoker;
import edu.indiana.extreme.lead.types.*;
import edu.indiana.dde.mylead.agent.MyLeadAgentInterface;

public class ClientTest {

    private MyLeadAgentInterface myleadAgentService;

    public ClientTest(String service_url) throws Exception{
        initServiceClient(service_url);
    }

    public void initServiceClient(String service_url) throws Exception{
        myleadAgentService =
            (MyLeadAgentInterface)ServiceInvoker.
                createXBeansRpcProxy(service_url,MyLeadAgentInterface.class);
    }

    public void sessiontest(String uID, String pid,
        MyLeadExperimentDocument expdoc,
        LeadEntryDocument datasourcedoc,
        LeadEntryDocument datastoredoc,
        LeadEntryDocument templatedoc,
        LeadEntryDocument instancedoc) {
```

```
try{

    /**
     * STEP 2-1: CREATE EXPERIMENT
     * create an experiment
     * createExperiment() returns the topicID for
     * notification/eventing service.
     * if you try to create an experiment under same name
     * it will throw an exception.
     */

    String topicId = myleadAgentService.createExperiment(uID,pid,expdoc);

    System.out.println("New Session is created in MyLEAD with topic id of "+topicId);

    /**
     * STEP 2-2: SET WORKFFLOW TEMPLATE
     * workflow template is stored under project. You have to
     * select which workflow template you will use for this experiment
     */

    myleadAgentService.setWorkflowTemplate(uID,pid,
                                           expdoc.getMyLeadExperiment().getName(),
                                           templatedoc);

    /**
     * STEP 2-3: SET WORKFLOW INSTANCE
     * After you create your workflow instance, you can store the
     * workflow instance in mylead under the experiment name.
     */

    myleadAgentService.setWorkflowInstance(uID,pid,
                                           expdoc.getMyLeadExperiment().getName(),
                                           instancedoc);

    /**
     * STEP 2-4: CREATE COLLECTION
     * MYLEAD 0.3 has two options to store files under collection.
     * You can store file under collection you defined, or collection automatically
     * generated for input file.
     * You can also generate nested collections.
     * Following line is generating an empty collection under the experiment.
     */
    // if the parent node is "", an empty collection will be created under exp.

    myleadAgentService.createCollection(uID,pid,
                                       expdoc.getMyLeadExperiment().getName(),
                                       "",
                                       "EXPERIMENT",
                                       "TESTCOLLECTION2005",
                                       "This is test collection.");

    // if the parent node is a collection, an empty collection will be created under
    // the parent collection.

    myleadAgentService.createCollection(uID,pid,expdoc.getMyLeadExperiment().getName(),
```

```

        "TESTCOLLECTION2005",
        "COLLECTION",
        "TESTCOLLECTION2005Coll",
        "This is test collection.");

/**
 * STEP 2-5: REGISTER FILE
 * MyLEAD 0.3 registers files under collection if you define collection name.
 * Also if the collection name is "", myLEAD will register under the default
 * input collection named as "[experiment name]:input data".
 */

// register two files under collection that defined earlier in this method.

myleadAgentService.registerFile(uID,pid,expdoc.getMyLeadExperiment().getName(),
    "TESTCOLLECTION2005", "filename2005",
    "testFile001-Desc",
    datasourcedoc);
myleadAgentService.registerFile(uID,pid,expdoc.getMyLeadExperiment().getName(),
    "TESTCOLLECTION2005", "filename2005",
    "testFile001-Desc",
    datasourcedoc);

// register two files under the default input collection

myleadAgentService.registerFile(uID,pid,expdoc.getMyLeadExperiment().getName(),
    "", "filename2005UnderInputDir-000",
    "testFile0-Desc",
    datasourcedoc);
myleadAgentService.registerFile(uID,pid,expdoc.getMyLeadExperiment().getName(),
    "", "filename2005-1UnderInputDir-001",
    "testFile1-Desc",
    datasourcedoc);

/**
 * STEP 2-6: QUERY ABOUT AN ACTIVE EXPERIMENT
 * Following examples using active experiment show how to query about
 * active experiment. These methods can be used for retrieving
 * information about currently running experiment. You can get simple
 * information such as template name or notification topic id associated with
 * current experiment. For more information, you can build a query using
 * these simple information and send it to mylead through
 * query APIs. After the experiment is finished, these information
 * are not available any more.
 */

System.out.println(myleadAgentService.
    getActiveWorkflowTemplateName(uID,
        pid,
        expdoc.getMyLeadExperiment().getName()
    ));

System.out.println(myleadAgentService.
    getActiveWorkflowInstanceName(uID,
        pid,
        expdoc.getMyLeadExperiment().getName()
    ));

```

```

        ));
        System.out.println(myleadAgentService.
            getCurrentState(uID,
                pid,
                expdoc.getMyLeadExperiment().getName()
            ));
        System.out.println(myleadAgentService.
            getActiveNotificationTopicID(uID,
                pid,
                expdoc.getMyLeadExperiment().getName()
            ));

/**
 * STEP 2-7: CLOSE ACTIVE EXPERIMENT
 * If you don't want to run workflow engine, it's OK. But you have to
 * close current active experiment manually. Otherwise, current experiment
 * will be remained in mylead agent and can cause problem later.
 * There are two types of closing APIs. closeExperiment() and shutdownExperiment().
 * If you have some request still running on mylead agent but you want the
 * result, please call closeExperiment(). It will deliver your query result and
 * close your experiment. Meanwhile, shutdownExperiment() will close your
 * experiment immediatly and delete all requests you sent earlier and didn't get
 * result yet.
 */

        myleadAgentService.shutdownExperiment(uID,
            pid,
            expdoc.getMyLeadExperiment().getName());

        System.out.println("session test end");
    } catch (Exception e) {
        e.printStackTrace();
    }
}

public void getProjects(String userid) throws Exception {
    MyLeadInvestigationArrayDocument a =
        myleadAgentService.getProjects(userid);
    System.out.println(a);
}

public void getWorkflowTemplate(String userid,
    String inv) throws Exception {
    LeadEntryArrayDocument a =
        myleadAgentService.getWorkflowTemplates(userid,
            inv,
            "",
            "");
    System.out.println("[getWorkflowTemplate]" + a.toString());
}

public void getExperiments(String userid,
    String inv) throws Exception {
    System.out.println("calling getExperiments");
    MyLeadExperimentArrayDocument a =

```

```
        myleadAgentService.getExperiments(userid,inv,3);
        System.out.println("[getExperiments]" + a);
    }

    public void getWorkflowInstance(String userid,
                                   String inv,
                                   String exp) throws Exception {
        LeadEntryArrayDocument a =
            myleadAgentService.getWorkflowInstance(userid,
                                                    inv,
                                                    exp,
                                                    "");
        System.out.println("[getWorkflowInstances]" + a.toString());
    }

    public void getWorkflowTemplateName(String userid,
                                        String inv) throws Exception {
        System.out.println("[getWorkflowTemplateName]" + myleadAgentService.
                            getWorkflowTemplateName(userid, inv).toString());
    }

    public void getNotifications(String userid,
                                 String inv,
                                 String exp,
                                 int count) {

        try {
            System.out.println(userid);
            System.out.println(inv);
            System.out.println(exp);
            System.out.println(count);

            WorkflowNotificationArrayDocument a =
                myleadAgentService.getNotifications(userid, inv, exp, count);
            System.out.println("[getNotifications]" + a.toString());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public void getActiveExperiment(String uid,
                                   String pid) {
        try {
            StringArrayDocument a =
                myleadAgentService.getActiveExperiment(uid, pid);
            StringArrayDocument sa =
                a.getStringArray1();
            for (int i = 0; i < sa.sizeOfSArray(); i++) {
                System.out.println(sa.getSArray(i));
            }
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

```
public void getActiveProject(String uid){
    try{
        StringArrayDocument a =
            myleadAgentService.getActiveProject(uid);
        StringArrayDocument.StringArray sa =
            a.getStringArray1();
        for (int i = 0; i<sa.sizeOfSArray(); i++){
            System.out.println(sa.getSArray(i));
        }
    }catch(Exception e){
        System.out.println(e);
    }
}

public void downloadMyWorkspace(String userid)
    throws Exception {

    System.out.println(myleadAgentService.getWorkspace(userid));
}

public void testCurrentExp(String userid,String pid, String exp)
    throws Exception {

    System.out.println(myleadAgentService.isActiveExperiment(userid,pid,exp));
    System.out.println(myleadAgentService.getActiveWorkflowTemplateName(userid,pid,exp));
    System.out.println(myleadAgentService.getActiveWorkflowInstanceName(userid,pid,exp));
    System.out.println(myleadAgentService.getCurrentState(userid,pid,exp));
    System.out.println(myleadAgentService.getActiveNotificationTopicID(userid,pid,exp));
}

public void addProject(String uid, String pid, String pdesc)
    throws Exception{
    myleadAgentService.createProject(uid, pid, pdesc);
    System.out.println("project is added");
}

public void addWorkflowTemplate(String uid,String pid,
                                LeadEntryDocument wfTemp)
    throws Exception{

    myleadAgentService.addWorkflowTemplate(uid,pid,wfTemp);

    System.out.println("Workflow Template is added");
}

public void doesUserHaveMyLeadAccount(String uid)
    throws Exception{
    System.out.println(myleadAgentService.
                        doesUserHaveMyLeadAccount(uid));
}

public static void main(String args[]) throws Exception {

    // PLEASE MAKE SURE YOU HAVE AN ACCOUNT IN MYLEAD SERVER
    String uid = "yourDN";
```

```
// PLEASE MAKE SURE MYLEAD AGENT SERVICE IS RUNNING
String serviceURL = "http://agent_service_URL";

String pid = "myTestProject2005";
String exp = "myTestExperiment2005Test";
String templatename = "myTEST_TemplateName";
String instancename = "myTEST_InstanceName";
String collectionname = "myTEST_CollectionName";
String filename = "myTEST_FileName";

MyLeadExperimentDocument expdoc = null;
LeadEntryDocument datasourcedoc = null;
LeadEntryDocument templatedoc = null;
LeadEntryDocument instancedoc = null;

try{
    // BE READY TO ACCESS MYLEAD AGENT.
    ClientTest ct = new ClientTest(serviceURL);

    // STEP0. Generate messages first.(You can generate message whenever you want.)
    expdoc =
        ClientTestMessageGenerator.generateExperiment(uid,pid,exp);

    templatedoc =
        ClientTestMessageGenerator.generateWorkflowTemplate(uid,pid,templatename);

    datasourcedoc =
        ClientTestMessageGenerator.generateDataSource(uid,pid,exp, "for demo");

    instancedoc =
        ClientTestMessageGenerator.generateWorkflowInstance(uid,pid,exp,instancename);

    System.out.println("#####"+
        " TEST START! #####");

    // STEP1. Create a project for your new experiments.
    //
    ct.addProject(uid,pid,"my test project 2005-0");

    ct.sessiontest(uid,pid,
        expdoc,
        datasourcedoc,
        null,
        templatedoc,
        instancedoc);

    ct.testCurrentExp(uid,pid,exp);

    // CHECK! downloadMyWorkspace will show your whole workspace.

    ct.downloadMyWorkspace(uid);

    ct.doesUserHaveMyLeadAccount(uid);
    // Query for your projects
    ct.getProjects(uid);

    // Query for your notification under your experiment
```

```
ct.getNotifications(uid,pid,exp,0);

// Query for your all experiments under the project
ct.getExperiments(uid,pid);

// Query for your template names under the project
ct.getWorkflowTemplateNameNames(uid,pid);

// ct.addWorkflowTemplate(uid, pid, templatedoc);
ct.getActiveExperiment(uid,pid);
ct.getActiveProject(uid);
// Query for your workflow instance under the experiment
ct.getWorkflowInstance(uid,pid,exp);

// Add a new workflow template under the project
ct.addWorkflowTemplate(uid,pid,templatedoc);

// Query for the workflow template you stored under the project
ct.getWorkflowTemplate(uid,pid);
ct.testCurrentExp(uid,pid,exp);
System.out.println("#####"+
                  " TEST END #####");
System.exit(0);
}catch(Exception e){
System.out.println("ClientTest: Problems beacuse of exception "+ e);
e.printStackTrace();
}
}
```