



# MyLEAD Publisher Release V1.3 Installation and Developer's Guide

Project/Release: LEAD / IU services release 3.0

Service: Publisher service V1.3

Document Title: myLEAD Publisher Installation and Developer's Guide

Organization: Indiana University

Date: August 4, 2007

Contact: Yiming Sun (yimsun@cs.indiana.edu)

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

[This page intentionally left blank]

---

1	Introduction.....	4
1.1	MyLEAD License .....	4
2	System Specification.....	5
2.1	Prerequisite Products.....	5
2.2	Required Libraries.....	5
3	Installing the MyLEAD Publisher Service .....	7
3.1	Installation.....	7
4	Configuration and Execution of the Publisher.....	8
4.1	Configuration of the Publisher's Properties File.....	8
4.1.1	URLs of other Core Services .....	8
4.1.2	Properties for Notification System.....	8
4.1.3	Properties for File Transfer.....	9
4.1.4	Miscellaneous Properties .....	9
4.2	Configuration of the Shell Script .....	9
4.3	Execution of the Publisher .....	9
5	Access to the MyLEAD Publisher Service.....	10
6	MyLEAD Publisher APIs .....	11
6.1	Working with XML Beans.....	11
6.2	Publishing an Object .....	11
7	Appendix.....	12
7.1	Publisher Service WSDL .....	12

[This page intentionally left blank]

## 1 Introduction

The myLEAD Publisher service allows its users to push data objects from the personal workspace to a community repository, and thus makes them available to others. This document provides instructions on the installation of this service and also serves as a guide to the developers accessing myLEAD Publisher service. The document covers:

- How to install and configure the service
- How to access the service from a client
- Description of the service APIs

### 1.1 MyLEAD License

The file `doc/myLEAD-License.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

The myLEAD Publisher service is designed to work on, and has been tested under, the following platforms:

- Redhat Linux

### 2.1 Prerequisite Products

Before you start developing a requester to the myLEAD Publisher service, the myLEAD server and the myLEAD Agent system should be installed and running properly. Below is a list of tools/services upon which the Publisher depends:

- JDK 1.5 or higher
- ANT 1.6.2 or higher
- myLEAD server 1.3.5
- myLEAD Agent 1.3.8
- Data Movement and Naming Service (DaMN)
- Data Catalog

For information about the installation of the myLEAD server and the myLEAD Agent system, please refer to their installation guides.

### 2.2 Required Libraries

Note: You can find all the required library jars in the myLEAD Publisher package. Individual jar files reside in subdirectories under the lib directory for better organization

- axis.jar
- cog-axis.jar
- cog-jglobus.jar
- commons-discovery.jar
- commons-logging.jar
- cryptix32.jar
- cryptix-asn1.jar
- damn\_services\_types-0.6.jar
- data\_catalog\_client-1.0.jar
- data\_catalog\_types-1.0.jar
- jaxrpc.jar
- jcd-jdk13-131.jar
- lead-metadata-1.8.jar
- leaddai-1.3.5.5.jar
- log4j-1.2.13.jar
- mylead-agent-typelib-1.3.8.jar
- ogsa.jar
- ogsa-activities.jar
- ogsadai-activities-Indiana2.jar
- ogsadai-activities-r6-syncStream.jar

- 
- ogsadai-core-r6-connectionPooling.jar
  - ogsadai-ogsi.jar
  - ogsadai-tools.jar
  - puretls-0.9b4.jar
  - saaj.jar
  - servlet.jar
  - workflow\_tracking-2.3.3.jar
  - workflow\_tracking\_types-2.3.3.jar
  - wsdl4j.jar
  - wsmg-1.76.7.jar
  - xalan.jar
  - xbean-2.2.0.jar
  - xercesImpl.jar
  - xmldb.jar
  - xmlParserAPIs.jar
  - xmlsec.jar
  - xpp3-1.1.3\_8.jar
  - xpp3\_xpath-1.1.3\_8.jar
  - xsul-2.10.4.jar
  - xsul\_xwsdlc-2.10.4.jar
  - xutil-0.2.jar

## 3 Installing the MyLEAD Publisher Service

### 3.1 Installation

The installation of the myLEAD Publisher service is easy – just unzip the distribution package to a directory. This document will refer to the installation directory as `$PublisherPath`.

After the package is successfully unzipped, it should contain at least the following files and directories. Please check to make sure they are all present for the successful operation of the Publisher service:

```
$PublisherPath> ls
  build.xml
  classpath.sh
  config/
  lib/
  LICENSE.txt
  log4j.properties
  publisher.properties
  run.sh
  src/
```

File attributes may or may not be retained by the distribution package, and therefore, it is recommended to do the following commands to make the scripts executable under Linux:

```
$PublisherPath> dos2unix *.sh
$PublisherPath> chmod 755 *.sh
```

The next step is to compile the Publisher's typelib, assuming Java and Ant are both installed:

```
$PublisherPath> ant generate
```

After this step, there should be a new directory called "generated"

```
$PublisherPath> ls
...
  generated/
...
```

Now it is time to compile the Publisher's service code:

```
$PublisherPath> ant
```

If the compilation is successful, there is yet another new directory called "build"

```
$PublisherPath> ls
  build/
...
```

## 4 Configuration and Execution of the Publisher

Before the Publisher service can be launched, there are a few files that must be configured according to the environment in which it was installed.

### 4.1 Configuration of the Publisher's Properties File

```
PUBLISHER_PROPERTY=PROPERTY_VALUE

PUBLISHER_PROPERTY_COUNT=n
PUBLISHER_PROPERTY_ARRAY1=PROPERTY_VALUE
...
PUBLISHER_PROPERTY_ARRAYn=PROPERTY_VALUE
```

The Publisher uses a few different services, and therefore, it relies on its properties file to configure how it interacts with them. A typical property of the Publisher can either be a name-value pair, or an array of name-value pairs. In the case of an array, there is always a property indicating the total number of elements to look for, and the array itself has unpadding index numbers directly appended to the end of the property names, and the index numbers start at 1. Sample formats of properties are shown above. The following sections will explain those properties by their groups in details.

#### 4.1.1 URLs of other Core Services

```
MYLEAD_SERVER_URL
MYLEAD_AGENT_WSDL_LOCATION
DATA_CATALOG_WSDL_LOCATION
DAMN_SERVICE_WSDL_LOCATION
```

The Publisher depends on three core services to function, which are myLEAD server, Data Catalog service, and Data Movement and Naming (DaMN) service. `MYLEAD_SERVER_URL` points to where myLEAD server resides, `DATA_CATALOG_WSDL_LOCATION` points to the WSDL location of the Data Catalog service, and `DAMN_SERVICE_WSDL_LOCATION` points to the WSDL location of the DaMN service.

In addition, the Publisher service uses the myLEAD Agent service to notify the outcome of the publish operation, and `MYLEAD_AGENT_WSDL_LOCATION` points to the WSDL location of the myLEAD Agent service.

#### 4.1.2 Properties for Notification System

```
NOTIFICATION_BROKER_URL
NOTIFICATION_MESSAGE_BOX_URL
NOTIFICATION_TOPIC_ID
```

The Publisher uses the messenger service and messagebox service to receive file transfer notifications from the DaMN service. `NOTIFICATION_BROKER_URL` and `NOTIFICATION_MESSAGE_BOX_URL` are properties to specify the URLs of the broker and the messagebox service, respectively. `NOTIFICATION_TOPIC_ID` specifies the topic under which the subscription is made.

### 4.1.3 Properties for File Transfer

```
PUBLISH_DESTINATION
PUBLISH_DESTINATION_TYPE
```

The Publisher service uses the DaMN services to copy users' personal data to a public repository. `PUBLISH_DESTINATION` defines the location of the public repository. In addition, there is also a `PUBLISH_DESTINATION_TYPE` property, which could have either a "manual" or an "auto" as its value. This is because the DaMN service supports two types of destinations. A "manual" destination type is to allow the user to specify exactly where a file should be transferred to; an "auto" destination is a logical name for a storage location known by the DaMN service, but the user does not need to know exactly where this storage is. An example of an auto destination would be "damn:iu:datacap", which tells the DaMN service to transfer a file to the Data Capacitor without concerning about where the data capacitor is physically located.

### 4.1.4 Miscellaneous Properties

```
LOG4J_PATH
```

The property `LOG4J_PATH` points to the `log4j.properties` file. The Publisher uses `log4j` to log useful information. For more details on configuration of `log4j`, please refer to `log4j`'s manual.

## 4.2 Configuration of the Shell Script

```
...
PATH_TO_PUBLISHER=$PublisherPath
...
JAVA_OPTS="-DMYLEAD_PUBLISHER_HOME=$PATH_TO_PUBLISHER
           -Dmylead.log4j.file=$PATH_TO_PUBLISHER/log4j.properties
           -Dcertskey=some/path/to/hostcertkey.pem
           -Dtrustedcerts=some/path/to/trusted_cas.pem
           $JAVA_OPTS"
...
```

Two lines in the shell script `run.sh` are likely to need configuration. `PATH_TO_PUBLISHER` should point to the installation directory of the Publisher. In addition, it is necessary to specify a few Java properties and pass them into the JVM as `JAVA_OPTS`.

`MYLEAD_PUBLISHER_HOME` is where the Publisher finds its home directory. The property `mylead.log4j.file` is used by the client jar to myLEAD server to find the `log4j.properties` file. In addition, it also passes in `certskey` and `trustedcerts` for communication using SSL over HTTP.

## 4.3 Execution of the Publisher

```
$PublisherPath> nohup ./run.sh publisher <port_number> &
```

To start the Publisher service, simply execute the command above and specify a port number on which the Publisher listens for incoming connections.

## 5 Access to the MyLEAD Publisher Service

First, please make sure you have a valid account on the myLEAD server. Also make sure that the myLEAD server, myLEAD Agent service, the Data Catalog service, the notification broker, the messagebox service, and the Data Movement and Naming service are accessible.

To access the myLEAD Publisher service, service requesters should include:

```
import xsul.XsulVersion;
import xsul.invoker.puretls.PuretlsInvoker;
import xsul.wsdl.WsdlResolver;
import xsul.wsif.spi.WSIFProviderManager;
import xsul.wsif_xsul_soap_gsi.Provider;
import xsul.xwsif_runtime.WSIFClient;
import xsul.xwsif_runtime.XmlBeansWSIFRuntime;
import edu.indiana.dde.mylead.publisher.MyLeadPublisherPortType;
```

Since most services are running using SSL for security, it is necessary to provide the Publisher client with a certificate file and a key file in order to contact the Publisher service. The paths to the files are usually passed into the client as Java properties:

```
String trustedCerts = System.getProperty("trustedcerts");
String certsKey = System.getProperty("certskey");

if (wsdlLocation.startsWith("https")) {
    PuretlsInvoker invoker =
        new PuretlsInvoker(certsKey, "", trustedCerts);

    Provider secureProvider = new Provider(invoker);

    WSIFProviderManager.getInstance().addProvider(secureProvider);

    WsdlResolver.getInstance().setSecureInvoker(invoker);
}
```

The myLEAD Publisher service is invoked by generating a client-side stub to the service. For example,

```
MyLeadPublisherPortType myLeadPublisherStub =
    (MyLeadPublisherPortType)XmlBeansWSIFRuntime.newClient(wsdlLocation).
        generateDynamicStub(MyLeadPublisherPortType.class);
```

## 6 MyLEAD Publisher APIs

### 6.1 Working with XML Beans

All inputs and outputs of a method are in the form of XML beans generated from the WSDL. While only XML Documents (e.g. `PublishObjectInputParamsDocument`) appear in the API signatures, a Document is merely a wrapper; the actual elements are always contained in the root element (e.g. `PublishObjectInputParamsType`) of that Document.

Consequently, throughout the following subsections, while the listed method signatures contain Documents, the detailed explanations on the methods will start with the root element. This discrepancy is intentional to avoid unnecessary complications.

### 6.2 Publishing an Object

```
public PublishObjectOutputParamsDocument  
    publishObject(PublishObjectInputParamsDocument input)
```

The myLEAD Publisher service v1.3 has only one API that is used exclusively for publishing a data object from a user's myLEAD workspace. The assumption is that the objects to be published exist in myLEAD workspace, and are also known by the DaMN service. Please see the WSDL listed in Appendix 7.1 for the exact parameter definitions

The input to this method is a `PublishObjectInputParamsType` object, which contains the following elements:

- a valid user DN
- LEAD resource ID of the object to be published
- the type of the object (PROJECT, EXPERIMENT, COLLECTION, or FILE)

This method returns a `PublishObjectOutputParamsType` object, which contains one of the following elements:

- a status string
- fault message

If the operation was successful, a status string with resource ID of the object to be published is returned to indicate the publish request has been received successfully; otherwise, the fault message contains information on the exception.

Please note that the publish operation is an asynchronous process, so that a successful return from the method invocation merely indicates the request has been received successfully. When the operation is finished, either successfully or failed, the Publisher contacts the myLEAD Agent service and sends a notification message indicating the outcome.

## 7 Appendix

### 7.1 Publisher Service WSDL

```
<?xml version="1.0" encoding="UTF-8"?>

<definitions name="myleadpublisher"
  targetNamespace="http://www.cs.indiana.edu/dde/namespace/mylead/publisher/0.1"
  xmlns:typens="http://www.cs.indiana.edu/dde/namespace/mylead/publisher/0.1/xsd"
  xmlns:wsdlns="http://www.cs.indiana.edu/dde/namespace/mylead/publisher/0.1"
  xmlns:xcatPar="http://www.dde.indiana.edu/namespaces/2004/10/xfw/xcat/parameter"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns="http://schemas.xmlsoap.org/wsdl/">

  <message name="MoveObject_RequestMessage">
    <part name="MoveObject_Run_InputParameters" element="typens:MoveObject_InputParams"/>
  </message>
  <message name="MoveObject_ResponseMessage">
    <part name="MoveObject_Run_OutputParameters" element="typens:MoveObject_OutputParams"/>
  </message>

  <message name="PublishObject_RequestMessage">
    <part name="PublishObject_Run_InputParameters"
      element="typens:PublishObject_InputParams"/>
  </message>
  <message name="PublishObject_ResponseMessage">
    <part name="PublishObject_Run_OutputParameters"
      element="typens:PublishObject_OutputParams"/>
  </message>

  <portType name="MyLeadPublisher_PortType">
    <operation name="publishObject">
      <input name="PublishObject_RequestMessage"
        message="wsdlns:PublishObject_RequestMessage"/>
      <output name="PublishObject_ResponseMessage"
        message="wsdlns:PublishObject_ResponseMessage"/>
    </operation>
  </portType>

  <binding name="MyLeadPublisher_SoapBinding" type="wsdlns:MyLeadPublisher_PortType">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="publishObject">
      <soap:operation soapAction=""/>
      <input name="PublishObject_RequestMessage">
        <soap:body use="literal"
          namespace="http://www.cs.indiana.edu/dde/namespace/mylead/publisher/0.1"
          required="true"/>
      </input>
      <output name="PublishObject_ResponseMessage">
        <soap:body use="literal"
          namespace="http://www.cs.indiana.edu/dde/namespace/mylead/publisher/0.1"
          required="true"/>
      </output>
    </operation>
  </binding>

  <service name="MyLeadPublisher">
    <port name="MyLeadPublisher_Port" binding="wsdlns:MyLeadPublisher_SoapBinding">
      <soap:address location="http://host.port/location"/>
    </port>
  </service>

  <types>
    <schema elementFormDefault="unqualified"
```

```
targetNamespace="http://www.cs.indiana.edu/dde/namespace/mylead/publisher/0.1/xsd"
  xmlns="http://www.w3.org/2001/XMLSchema"
  <annotation>
    <documentation xml:lang="en">
      This XML schema is for MyLeadPublisher
    </documentation>
  </annotation>

  <element name="PublishObject_InputParams"
    type="typens:PublishObject_InputParamsType"/>
  <element name="PublishObject_OutputParams"
    type="typens:PublishObject_OutputParamsType"/>

  <complexType name="PublishObject_InputParamsType">
    <sequence>
      <element name="DN" type="string" minOccurs="1" maxOccurs="1">
        <annotation>
          <documentation xml:lang="en">
            DN of the original owner of the object to be published.
          </documentation>
        </annotation>
      </element>
      <element name="resourceID" type="string" minOccurs="1" maxOccurs="1">
        <annotation>
          <documentation xml:lang="en">
            LEAD resourceID of the object to be published.
          </documentation>
        </annotation>
      </element>
      <element name="type" type="string" minOccurs="1" maxOccurs="1">
        <annotation>
          <documentation xml:lang="en">
            LEAD resource type
          </documentation>
        </annotation>
      </element>
    </sequence>
  </complexType>

  <complexType name="PublishObject_OutputParamsType">
    <sequence>
      <element name="results" type="string">
        <annotation>
          <documentation xml:lang="en">
            Results of the publish operation.
          </documentation>
        </annotation>
      </element>
      <element name="faultMessage" type="string">
        <annotation>
          <documentation xml:lang="en">
            FaultMessage
          </documentation>
        </annotation>
      </element>
    </sequence>
  </complexType>
</schema>
</types>
</definitions>
```