



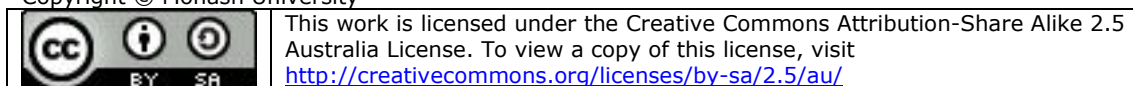
JAHDL v 1.1

Technical Reference Manual

Release History

Release	Date (yyyy-mm-dd)	Author/ Reviewer	Description	Release identifier
V1.1	2007-08-29	AC	Initial release	hdl: 102.100.272/6QJD6N6QH

Copyright © Monash University



This work is created as part of the PILIN – Persistent Identifier Linking Infrastructure project. The PILIN project is sponsored by the Australian Commonwealth Department of Education, Science and Training under the ARROW Project.

JAHDL v1.1 1

Technical Reference Manual..... 1

1. Scope..... 3

2. What is JAHDL?..... 3

3. Integrating JAHDL handle operations 4

Handle server authentication data 5

Handle meta data..... 5

4. Single handle CUD operations using XML request File 6

5. Batch handle CUD operations using XML request File 8

6. Single handle CUD operations using request object..... 10

7. Batch handle CUD operations using request object..... 13

8. Handle resolution operation [currently available for single handle only] 14

9. JAHDL Handle Management Tool..... 16

10. JAHDL based systems..... 16

11. Contacts..... 16



1. Scope

This scope of this document is to help developers integrate JAHDL handle operations into their application.

It is intended to be used as JAHDL Technical Reference.

2. What is JAHDL?

JAHDL is Java API for Handle. It is a free open source software governed by MOZILLA PUBLIC LICENSE version 1.1

JAHDL requires Java 1.5 or above.

It is available for download from

<http://pilin.net.au/pilin/downloads/jahdl.html>

It performs single create, resolve, update & delete and also batch create, update & delete handle operations.

See <http://www.handle.net> for handle definition

It is built on top of the CNRI handle java libraries available from the same web site.

JAHDL is an extension layer which extends CNRI handle library capabilities by transforming some industry standard inputs [e.g XML file] into the required handle meta data format, thus making it a easier to integrate handle operations within other applications.

Non java applications can integrate with JAHDL by utilising the command line handle management tool available. For details see section 'JAHDL Handle Management Tool'.

JAHDL also removes some complexities involved in creating handles and provides some default selection criteria e.g. Each time JAHDL is used for creating a handle, users need not provide extra meta data for HS_ADMIN field [handle administration permissions + other meta data] as JAHDL adds a default HS_ADMIN field computed from the handle meta data available.

This feature can be turned off by providing a user defined handle admin meta data. In most cases, this will not be required.

For more details see tutorial section below.

3. Integrating JAHDL handle operations

JAHDL is architected in a service oriented manner i.e. each handle operation is managed by an individual service, dedicated to the operation, taking requests and returning a response.

Consequently the current release of JAHDL has defined the following 7 services available for consumption:

- Single handle create service
- Single handle update service
- Single handle delete service
- Single handle resolution service
- Batch handle create service
- Batch handle update service
- Batch handle delete service

Each service can be consumed in two ways:

- By providing handle meta data in an XML file, or
- By providing handle meta data encapsulated in java objects

The above model is implemented technically by providing two overloaded methods in each service interface.

All the services require request data to perform an operation. JAHDL's service request data can be classified broadly into two categories:

- Handle server authentication data
- Handle meta data

Handle server authentication data

JAHDL doesn't provide any security model other than the default PKI security model implemented by the CNRI handle libraries. In order to perform a handle operation, a JAHDL user has to provide the following two credentials, henceforth referred to as Handle server authentication data:

- Handle server private key. [admpriv.bin file available in handle server local installation directory]
- Private Key passphrase, if any, used to decrypt the key. This is the same passphrase used for starting and stopping the handle server, if required.

The above credentials are not copied/stored during/after the handle operation by JAHDL.

Handle meta data

Handle meta data comprises of different handle values data used for constructing a handle. Details about the handle technology is available at www.handle.net

Let us now look at some code snippets for consuming JAHDL services.

Note: Most of the code snippets are taken from the sample JAHDL clients shipped with JAHDL distribution [open source software]. For more details visit www.pilin.net.au

4. Single handle CUD operations using XML request File

All the handle operation job request XML files must conform to PILIN job request schema.

Some sample PILIN handle operation request files and the PILIN schema are available in JAHDL.

Download JAHDL from <http://pilin.net.au/pilin/downloads/jahdl.html>

JAHDL basically requires three pieces of information for any single or batch CUD handle operation - Handle server authentication data[server private key + key passphrase] and the handle meta data already encapsulated in the XML operation request file.

First of all we need to convert the server private key and the XML operation request data into byte array, as required by the services.

```
// GENERATE REQUEST DATA
byte[] serverPrivateKey =
au.net.pilin.jahdl.util.core.PilinUtil.getFileAsBytes(serverKeyLocation);
//e.g. /hs/serv_1/bin/admpriv.bin or C:/handle/server/admpriv.bin

byte[] xmlRequest =
au.net.pilin.jahdl.util.core.PilinUtil.getFileAsBytes(xmlRequestFileLocation);
//e.g. /usr/loc/handle/request/create-handle.XML

//CONSUME JAHDL SERVICE
// init service
HandleCreateService service = new HandleCreateServiceImpl();
String handleId = null;
try
{
// execute createHandle method.
handleId = service.createHandle(xmlRequest, serverPrivateKey, keyPassphrase);
System.out.println("handle " + handleId + " created.");
}
catch (RequestFileParserException e)
{
//exception thrown if errors in XML parsing
System.err.println("XML parsing exception: " + e.getMessage());
System.exit(-1);
}
catch (HandlePrivateKeyException e)
{
//exception thrown if corrupted private key or incorrect passphrase
System.err.println("Handle server private key error:" + e.getMessage());
System.exit(-1);
}
}
```

```
catch (HandleRequestDataException e)
{
//exception thrown if handle operation request data is incomplete
System.err.println("Incomplete handle meta data: " + e.getMessage());
System.exit(-1);
}
catch (HandleCreateException e)
{
//exception thrown if error is system level
System.err.println("Handle could not be created: " + e.getMessage());
System.exit(-1);
}
```

As seen above the service returns the complete handle id if the operation is successful. If there is an error then based on the error category a suitable exception is thrown.

For a single handle UD operation, the desired service should be initialised and then consumed in a similar way. All the services throw similar categories of exceptions.

```
//for single handle update operation
HandleUpdateService service = new HandleUpdateServiceImpl();
service.updateHandle(xmlRequest, serverPrivateKey, keyPassphrase);
```

```
//for single handle delete operation
HandleDeleteService service = new HandleDeleteServiceImpl();
service.deleteHandle(xmlRequest, serverPrivateKey, keyPassphrase);
```

5. Batch handle CUD operations using XML request File

Batch handle CUD operations using XML request file have a very similar approach to single handle operations discussed above, the difference being in the service response.

When consuming batch operations the services will return `PilinBatchOperationResponse` object which contains the batch operation log and a flag to specify if at least one of the batch operations has failed.

Note: Batch operations do not implement ALL or NONE policy thus there are always chances of subset of the batch not being affected. So always check the flag to make sure there are no undesired results.

```
// GENERATE REQUEST DATA
byte[] serverPrivateKey =
au.net.pilin.jahdl.util.core.PilinUtil.getFileAsBytes(serverKeyLocation);
byte[] xmlRequest =
au.net.pilin.jahdl.util.core.PilinUtil.getFileAsBytes(xmlRequestFileLocation);

// INIT LOGGER FOR FLUSHING OPERATION LOG
PrintWriter logger = new PrintWriter(new FileWriter(logFileLocation), true);

//CONSUME JAHDL SERVICE
BatchHandleCreateService service = new BatchHandleCreateServiceImpl();
PilinBatchOperationResponse response = null;
try
{
// execute createBatchHandles method.
response = service.createBatchHandles(xmlFile, handleServerKey,
keyPassphrase);
// flush log
logger.write(new String(response.getBatchOperationLog()));
if (response.getAtLeastOneBatchOperationFailed())
{
System.out.println("At least one handle operation failed. View details at: " +
logFileLocation);
}
else
{
System.out.println("Batch operation successfull. View details at: " +
logFileLocation);
}
}
```

```
}  
catch (RequestFileParserException e)  
{  
System.err.println("XML parsing exception: " + e.getMessage());  
System.exit(-1);  
}  
catch (HandlePrivateKeyException e)  
{  
System.err.println("Handle server private key error: " + e.getMessage());  
System.exit(-1);  
}  
catch (HandleRequestDataException e)  
{  
System.err.println("Incomplete handle meta data: " + e.getMessage());  
System.exit(-1);  
}  
catch (BatchHandleCreateException e)  
{  
System.err.println("Batch handles could not be created: " + e.getMessage());  
System.exit(-1);  
}  
finally  
{  
logger.flush();  
logger.close();  
}
```

As seen above the flag 'atLeastOneBatchOperationFailed' is checked to make sure there are no failures. Similarly for batch handle UD operation, the desired service should be initialised and then consumed in a similar way.

All the services throw similar categories of exceptions.

```
//for batch handle update operation  
BatchHandleUpdateService service = new BatchHandleUpdateServiceImpl();  
response = service.createBatchHandles(xmlRequest, serverPrivateKey,  
keyPassphrase);
```

```
//for batch handle delete operation  
BatchHandleDeleteService service = new BatchHandleDeleteService();  
response = service.deleteBatchHandles(xmlRequest, serverPrivateKey,  
keyPassphrase);
```

6. Single handle CUD operations using request object

This overloaded functionality provides another channel to integrate handle operations. Both the overloaded functionalities share common business logic and return same results. Developers interested in adding handle operations to their work flow will find this section useful. First create a request object by populating it with Handle server authentication data + handle meta data.

Later this data will be passed to the service for consumption.

```
// CREATE PilinHandleRequest object
PilinHandleRequest pilinHandleRequest = new PilinHandleRequest();

//SET HANDLE SERVER AUTHENTICATION DATE
// set handle server private key
pilinHandleRequest.setHandleServerKey(handleServerKeyLocation);
// set certificate passphrase, if any
pilinHandleRequest.setKeyPassphrase(keyPassphrase);
```

At this stage the request object as been populated with Handle server authentication data.

Handle meta data shall be populated now.

```
//SET HANDLE NAMESPACE
pilinHandleRequest.setHandleNamespace(handleNamespace);

//SET HANDLE SUFFIX
pilinHandleRequest.setHandleSuffix(handleSuffix);
```

Handle namespace and suffix have been defined and the handle values now need to be defined.

Every handle should have a HS_ADMIN type handle value. This value is useful for administering the handle.

BY default JAHDL will add a default HS_ADMIN value with namespace as 'handleNamespace' and authorising index as 300. In some cases there may be a need to specify a different admin namespace. This can be done as shown below:

```
//SET HS_ADMIN AS DESIRED
```

```
int adminAuthorisingIndex = 300;
pilinHandleRequest.setHandleAdminWithAllPermissions(adminNamespace,
adminAuthorisingIndex);
```

The above code snippet adds HS_ADMIN with all permissions. Each HS_ADMIN field is associated with 12 different permissions.

The default is set to true for all.

In case you need to override with specific permissions do so as set out set as below:

```
//SET HS_ADMIN AS DESIRED
int adminAuthorisingIndex = 300;
pilinHandleRequest.setHandleAdmin(adminNamespace, adminAuthorisingIndex,
canAddHandle, canDeleteHandle,
canAddNA, canDeleteNA, canModifyValue, canRemoveValue, canAddValue,
canReadValue, canModifyAdmin, canRemoveAdmin, canAddAdmin,
canListHandles);
```

Each handle value has an index, a type, a value along with other meta data like permissions, timestamp, TTL, etc...

JAHDL allows the user to create a handle value having their own customised type and desired index.

A handle value is a collection of meta data and JAHDL hides some complexities by providing some default behaviour.

If an index is not specified for a handle value JAHDL will try to index it using its own reference index list.

[au.net.pilin.jahdl.util.core.PilinHandleIndex]

If the type is not predefined in the index list [e.g. your own customised type] then it becomes mandatory to supply an index value, otherwise an exception will be thrown.

Each handle value is administered by a set of four permissions.

As before, you will find a method to set default permissions or you can set your own specific ones.

Let's add some other common handle values now.

Here we want to add a handle value of type URL and specify its value to be say 'www.dummyURL.com.au'

```
// ADD HANDLE VALUE OF TYPE URL
pilinHandleRequest.setHandleValueWithDefaultPermissions(PilinHandleType.URL,
'www.dummyURL.com.au');
```

The utility class au.net.pilin.jahdl.util.core.PilinHandleType is available for referencing to registered handle types.

To add desired permission use as below:

```
// ADD HANDLE VALUE OF TYPE URL WITH DESIRED PERMISSIONS
pilinHandleRequest.setHandleValue(PilinHandleType.URL,
'www.dummyURL.com.au', adminCanRead, adminCanWrite, publicCanRead,
publicCanWrite);
```

Setting default permissions will set all as true expect publicCanWrite permission.

The above URL handle value will have its index set referenced from PilinHandleIndex.

To set your desired index use as below:

```
// ADD HANDLE VALUE OF TYPE EMAIL WITH DESIRED INDEX
int desiredIndex = 666;
pilinHandleRequest.setHandleValueWithDefaultPermissions(desiredIndex,
PilinHandleType.EMAIL, 'dummy@dummyURL.com.au');
```

Multiple handle values can be added for the same type. Just give a different index for each value, if setting index manually.

```
// ADD MULTIPLE HANDLE VALUES OF SAME TYPE
pilinHandleRequest.setHandleValueWithDefaultPermissions(666,
PilinHandleType.EMAIL, 'dummy1@dummyURL.com.au');
pilinHandleRequest.setHandleValueWithDefaultPermissions(999,
PilinHandleType.EMAIL, 'dummy2@dummyURL.com.au');
```

The following code snippet shows how to create a handle value with customised type.

```
// ADD HANDLE VALUE OF USER DEFINED TYPE
pilinHandleRequest.setHandleValueWithDefaultPermissions(userDefinedType,
'stringValue');
```

```
//SET A HANDLE INDEX TO BE DELETED WHILE UPDATING HANDLE
//use any of the following methods to mark handle index for
deletion.
```

```
req.setHandleIndexListToDelete(int[])
req.setHandleIndexListToDelete(Integer[])
req.setHandleIndexListToDelete(List<Integer>);
```

```
//SET THE HANDLE VALUE TO BE APPENDED WHILE UPDATING.
//use any of the following methods to append values for an existing
handle.
```

```
req.setHandleValuesToAppend(List<PilinHandleValue>;
handleValuesToAppend);
```

```
req.setHandleValueToAppendWithDefaultPermissions(PilinHandleType.EMAI
L, Helper.DUMMY_HANDLE_OWNER_EMAIL_UPDATED);
```

```
req.setHandleValueToAppend(PilinHandleType.URN,
Helper.DUMMY_URN_UPDATED, true, true, true, false);
```

```
req.setHandleValueToAppendWithDefaultPermissions(Helper.DUMMY_DESC_INDEX, PilinHandleType.DESC, Helper.DUMMY_DESC_UPDATED);
```

```
req.setHandleValueToAppend(Helper.DUMMY_HS_SERV_INDEX, PilinHandleType.HS_SERV, Helper.DUMMY_HS_SERV, true, true, true, false);
```

Consume the service now.

```
//CONSUME JAHDL SERVICE  
HandleCreateService service = new HandleCreateServiceImpl();  
String handleID = service.createHandle(pilinHandleRequest);  
System.out.println("Handle " + handleID + " created.");
```

Similar category exceptions are thrown as discussed before. The process for handle update operation populating the request data is similar. Set the handle values you want to update and consume the service.

In the case of delete handle service only set the handle namespace and suffix along with handle server authentication data to delete the handle.

7. Batch handle CUD operations using request object

Batch handle operation services requires `PilinBatchHandleRequest` object.

`PilinBatchHandleRequest` holds collection of `PilinHandleRequest` object wherein each `PilinHandleRequest` object maps to a single handle in the batch.

Populate individual `PilinHandleRequest` object as explained above and add the List to `PilinBatchHandleRequest`.

Just make sure to add handle server authentication data to the `PilinBatchHandleRequest` object only. No need to add the same information to each individual `PilinHandleRequest` object as you

would do in single handle operation. Then consume services in a similar way as defined above.

8. Handle resolution operation [currently available for single handle only]

Handle resolution service provides resolution of handles. It takes a HandleResolverRequest object process it and returns a HandleResolverResponse object.

The user needs to specify either handle namespace and suffix separately or you can assign complete handle id to the request object.

```
// CREATE HandleResolverRequest object
HandleResolverRequest resolverRequest = new HandleResolverRequest();
resolverRequest.setHandleId(completeHandleId);
```

Passing this request object to HandleResolverService will result in returning all the handle values assigned to the handle.

Resolver service returns HandleResolverResponse object as a response. This response object contains a collection of PilinHandleValue which in turn is handle value meta data holder class.

If a user wants to filter out the resolution meta data, they can provide either the handle types or handle indexes or both.

```
//SET RESPONSE TO CONTAIN URL AND EMAIL TYPE
resolverRequest.setHandleType(PilinHandleType.URL);
resolverRequest.setHandleType(PilinHandleType.EMAIL);
```

```
//SET RESPONSE TO CONTAIN HANDLE VALUES WITH INDEX 666 and 999
resolverRequest.setHandleIndex(666);
resolverRequest.setHandleIndex(999);
```

The above resolution request will notify the resolver service to include in the response result only the specified types and indexes.

The request is now processed.

```
// CONSUME JAHDL SERVICE
HandleResolverService service = new HandleResolverServiceImpl();
HandleResolverResponse resolverResponse = service.resolveHandle(new
HandleResolverRequest());
```

```
//ITERATE THROUGH HANDLE VALUES AND DISPLAY
PilinHandleValue[] handleValues = resolverResponse.getHandleValues();
for (int x = 0; x < handleValues.length; x++)
{
```

```
    PilinHandleValue handleVal = handleValues[x];
    System.out.println("handle val " + x + " type:" + handleVal.getType());
    System.out.println("handle val " + x + " index:" + handleVal.getIndex());
    System.out.println("handle val " + x + " value:" + handleVal.getValue());
    System.out.println("handle val " + x + " timestamp:" +
    handleVal.getTimestampAsString());
    System.out.println("handle val " + x + " TTL:" + handleVal.getTTL());
    System.out.println("handle val " + x + " adminCanRead permission:" +
    handleVal.getAdminCanRead());
    System.out.println("handle val " + x + " adminCanWrite permission:" +
    handleVal.getAdminCanWrite());
    System.out.println("handle val " + x + " publicCanRead permission:" +
    handleVal.getPublicCanRead());
    System.out.println("handle val " + x + " publicCanWrite permission:" +
    handleVal.getPublicCanWrite());
}
```

9. JAHDL Handle Management Tool

JAHDL provides a useful utility class `au.net.pilin.jahdl.util.core.PilinHandleMgmtTool`. This is a command line utility which takes 5 command line arguments:

1. Handle server private key location,
2. Private key passphrase if any,
3. Handle operation request XML file,
4. Handle operation. Valid handle operations are 'create-handle' 'update-handle' 'delete-handle' 'batch-create-handle' 'batch-update-handle' 'batch-delete-handle'
5. Location of log file where JAHDL will flush the operation logs.

Currently users can integrate their non java platform applications by utilising this command line utility and invoking it in a unix shell script or windows batch file like environment.

Example calling from command line:

```
$ java -jar jahdl_1.1.jar <handle server private key location> <key  
passphrase> <XML request file location> <handle operation> <log  
file location>
```

10. JAHDL based systems

- There is a demo on-line which is built on top of JAHDL libraries to expose JAHDL functionalities to http world. For details see <http://pilin.net.au>
- PILIN development team has also exposed these services as web services. For details see <http://pilin.net.au>

11. Contacts

Feedback: feedback-jahdl@pilin.net.au

Technical Assistance: amit.chaudhary@lib.monash.edu.au