



# Persistent Identifier Service

24 November 2008

Last Updated: 24 June 2010

1. Purpose .....	2
2. Overview .....	2
3. Services .....	3
4. Handle Resolution.....	6
5. Current Clients .....	6
6. Recommended Usage .....	7
7. Example Scenario .....	7

## 1. Purpose

The purpose of this document is to provide an overview of the Persistent Identifier Service (PIDS) software developed under the Australian National Data Service (ANDS) Establishment Program.

## 2. Overview

The Persistent Identifier Service (PIDS) was not designed as a software package for end users. It does not come packaged with a friendly user interface, rather it is intended it be used for integration with other applications requiring a persistent identifier service.

At an implementation level, PIDS is a Tomcat web application with each service point being implemented as a Java servlet. A Postgres database provides the storage layer using the default CNRI Handle Server schema (as described in the Handle Technical Overview document). To simplify authentication of PIDS to the Local Handle Server (LHS) and keep these communications secure, PIDS must reside on the same machine as the LHS to take advantage of the default PKI authentication. Authentication of clients is required before access to administrative services is granted. This is currently implemented by registering trusted clients in the Postgres database along with a unique key issued by the Handle Service. This key coupled with the client's identity (IP address) is used in determining whether a request to a service is granted.

Service calls are typically HTTP requests generated via a user interface or automated process containing an XML document. The XML document provides the identity of requestor, typically the owner of the target handle. This information is used to ensure the requestor has sufficient rights to undertake the administrative operation on the target handle. As an example, in the XML fragment below the **applID** is the Service-issued key identifying a PIDS trusted client, the **identifier** is the handle owner and **authDomain** specifies the authentication domain in which the user is operating (such as Shibboleth, LDAP, or some other trusted domain).

```
<request name="addValue">
  <properties>
    <property name="appld" value="5d9a4da3580c528ba98d8e6f088dab93f680dd6b" />
    <property name="identifier" value="scott" />
    <property name="authDomain" value="mycomputer.edu.au" />
  </properties>
</request>
```

Note that the owner of the handle can only manage their handles from within the **authDomain** the handle was minted. The **user** and **authDomain** cannot be updated via service calls at this point in time. The implication of this is a user moving from one authentication domain to another (for example a different institution or application) will not be able to manage their handles without first organising an updating of their details with the PIDS service manager. The owner assigned when a handle is minted is the only user able to undertake further administrative operations on the handle.

Responses to service calls are also in an XML form and vary depending on the service called and whether the service call succeeded or failed. Requests based on a single handle will return the state of the handle **after** a successful request.

Each handle in the Identifier Service has two administrator records by default, namely the Handle Server admin (an HS\_ADMIN entry) as well as the owner (a custom handle value

type, AGENTID), i.e. the individual either directly creating the handle or the automated agent (application) on whose behalf the handle was created. An owner is assigned a handle and this value appears in the AGENTID value of an individual owner's handle records.

### 3. Services

#### 3a. Administrative Services

##### Service: mint

###### Parameters:

type (Mandatory)

Must be either empty, "DESC" or "URL"

value (Mandatory)

Must be either empty, contain brief descriptive text (type=DESC) or a URL (type=URL)

index (Optional)

A numeric value indicating the index at which to store the value

**Description:** Responsible for the creation of a handle. If **type** and **value** are both empty a handle with no values is created. The handle is assigned an HS\_ADMIN value at index 100 representing the handle server administrator and an AGENTID value at index 101 containing the handle of the owner of the newly created handle. If the owner is not known to the handle system, an owner is created from the **identifier** and **authDomain** values from the request and assigned a handle. This owner handle is also treated the same as a newly minted handle with the addition of a non-publicly readable DESC value at index 102 describing the owner.

###### Example response:

```
<?xml version="1.0" encoding="UTF-8"?>
<response type="success">
  <identifier handle="10378.2/12">
    <property index="1" type="URL" value="http://ands.org.au"/>
  </identifier>
  <timestamp>2009-05-15T01:41:11Z</timestamp>
  <message type="user">Successfully authenticated and created handle</message>
</response>
```

##### Service: addValue

###### Parameters:

handle (Mandatory)

The handle to which the value is to be added

type (Mandatory)

Must be either "DESC" or "URL"

value (Mandatory)

The new value

**Description:** Adds a value to an existing handle. Only the owner of the handle is able to add a value to it and only types of DESC and URL are able to be added.

###### Example response:

```
<?xml version="1.0" encoding="UTF-8"?>
<response type="success">
  <identifier handle="10378.2/12">
    <property index="1" type="URL" value="http://ands.org.au"/>
  </identifier>
</response>
```

```
<property index="2" type="DESC" value="ANDS Home Page"/>
</identifier>
<timestamp>2009-05-15T02:14:45Z</timestamp>
<message type="user">Successfully updated handle</message>
</response>
```

**Service: addValueByIndex**

**Parameters:**

handle (Mandatory)

The handle to which the value is to be added

type (Mandatory)

Must be either "DESC" or "URL"

value (Mandatory)

The new value

index (Mandatory)

A numeric value indicating the index at which to store the value

**Description:** Adds a value to an existing handle. Only the owner of the handle is able to add a value to it and only types of DESC and URL are able to be added.

**Example response:**

```
<?xml version="1.0" encoding="UTF-8"?>
<response type="success">
  <identifier handle="10378.2/12">
    <property index="1" type="URL" value="http://ands.org.au"/>
    <property index="2" type="DESC" value="ANDS Home Page"/>
  </identifier>
  <timestamp>2009-05-15T02:14:45Z</timestamp>
  <message type="user">Successfully updated handle</message>
</response>
```

**Service: modifyValueByIndex**

**Parameters:**

handle (Mandatory)

The handle that is to have one of its values modified

index (Mandatory)

The index of the value to modify

value (Mandatory)

The new value

**Description:** Changes a value associated with an existing handle. Only the owner of the handle is able to modify a value and only types of DESC and URL are able to be modified. To obtain available indexes a call to the getHandle service will return all available values and their indexes.

**Example response:**

```
<?xml version="1.0" encoding="UTF-8"?>
<response type="success">
  <identifier handle="10378.2/12">
    <property index="1" type="URL" value="http://www.ands.org.au.au"/>
    <property index="2" type="DESC" value="ANDS Home Page"/>
  </identifier>
```

```
<timestamp>2009-05-15T02:23:01Z</timestamp>
<message type="user">Successfully modified handle value</message>
</response>
```

#### **Service: deleteValueByIndex**

##### **Parameters:**

handle (Mandatory)

The handle from which a value is to be deleted

index (Mandatory)

The index of the value to modify

**Description:** Deletes a value associated with an existing handle. Only the owner of the handle is able to delete a value and only types of DESC and URL can be deleted. To obtain available indexes a call to the getHandle service will return all available values and their indexes for a handle.

##### **Example response:**

```
<?xml version="1.0" encoding="UTF-8"?>
<response type="success">
  <identifier handle="10378.1/44">
    <property index="1" type="URL" value="http://www.and.s.org.au.au"/>
  </identifier>
  <timestamp>2009-05-15T02:29:27Z</timestamp>
  <message type="user">Successfully deleted handle value</message>
</response>
```

#### **Service: listHandles**

##### **Parameters:**

startHandle (Optional)

**Description:** Returns a list of handles owned by the requestor, an owner being a combination of identifier and authDomain passed in the body of the request. Note this service is intended for listing a small number of handles for the use case of serving a self-managed GUI environment. Therefore its response is limited in the number of handles returned. If there is a need for obtaining all handles owned by a particular user please contact the service maintenance agency to discuss requirements as a more targeted service may need to be implemented to meet requirements. Alternatively, this service can be repeatedly called using the **startHandle** parameter which should be set to the last listed handle from the previous **listHandles** call.

##### **Example response:**

```
<?xml version="1.0" encoding="UTF-8"?>
<response type="success">
  <identifiers>
    <identifier handle="10378.2/12"/>
    <identifier handle="10378.2/13"/>
    <identifier handle="10378.2/14"/>
    <identifier handle="10378.2/15"/>
  </identifiers>
  <timestamp>2009-05-15T02:03:59Z</timestamp>
  <message type="user">Handle listing successful</message>
</response>
```

### **3b. Non-administrative Services**

**Service: getHandle****Parameters:**

handle (Mandatory)

The handle whose details are to be retrieved

**Description:** Returns publicly readable handle values along with index and type information

**Example response:**

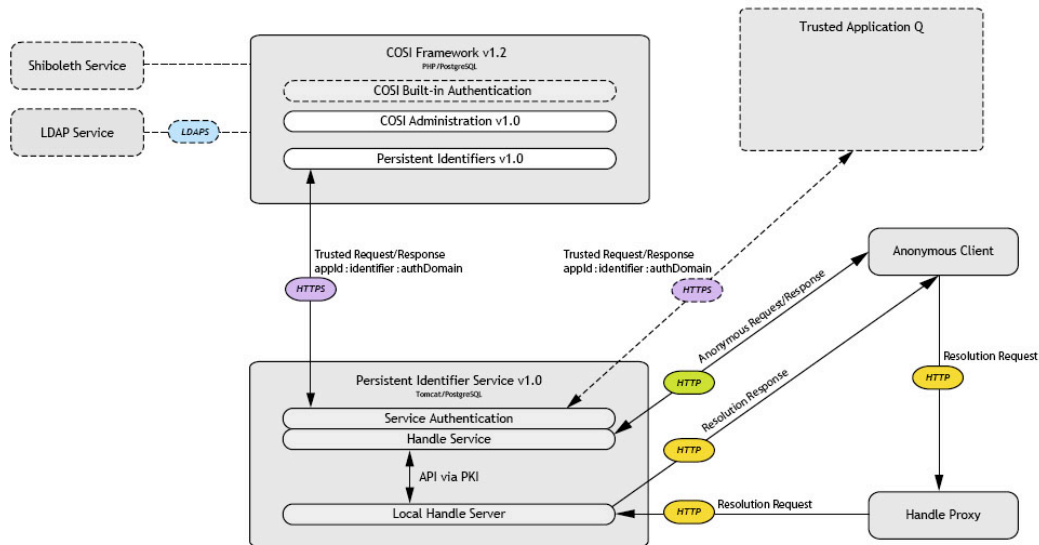
```
<?xml version="1.0" encoding="UTF-8"?>
<response type="success">
  <identifier handle="10378.2/12">
    <property index="1" type="URL" value="http://www.andis.org.au "/>
  </identifier>
  <timestamp>2009-05-15T03:15:13Z</timestamp>
  <message type="user">Request successful</message>
</response>
```

**4. Handle Resolution**

Handle resolution is undertaken by the existing CNRI Handle Proxy service located at <http://hdl.handle.net>. This service caches handle values to avoid latency and handle system traffic with the result that changes to handle values in the ANDS PID service are not immediately available by an unqualified reference to the Handle Proxy. The TTL (Time-To-Live) of all ANDS PIDS handle values is set at 30 minutes as per advice from CNRI. This means that any updates to handle values may not be cached in the Handle Proxy for at least 30 minutes after committed to the ANDS PID Service. The Handle Proxy has means for bypassing the proxy cache for critical applications if required by service users. Further information can be found at <http://www.handle.net/faq.html#4.15>

**5. Current Clients**

PIDS has a single trusted client application, Persistent Identifiers v1.0, implemented under the COSI framework which provides a user interface (UI) for self-service identifier management. The UI makes various HTTP service calls as described earlier to allow a user to create new handles as well as modify and list existing handles they own. Resolution is delegated to the CNRI Handle Proxy. The relationship between COSI and PIDS is shown in the architecture diagram below.



## 6. Recommended Usage

The service is primarily aimed at content management or similar applications wanting to support persistent identifiers without running their own PI service. It is assumed these applications will ensure they keep handle values updated in instances of content relocation or migration. The service while able to support manual management, is not designed around this assumption as the likelihood of individuals maintaining content locations (or other handle metadata) outside a managed environment seems remote.

It is strongly recommended that application developers ensure service outages can be catered for in their applications. While every attempt to ensure the service is running the reality is that unexpected outages will occur.

## 7. Example Scenario

Assume institution Org1 is developing a content management system (CMS) and wants to integrate the ANDS Persistent Identifier Service to obtain handles for identifying objects. Org1 would typically undertake the following process:

Request registration with the service

Obtain the appld to use in all administrative requests as described earlier

Develop service integration code and test the application against the test service

Request registration with the production service

The integration development required by the CMS developer for the above scenario may be as simple as minting a handle on deposit of content to the CMS and maintaining a single URL associated with that content over time. This requires use of the **mint** and **modifyValueByIndex** services. When content is ingested, the CMS using the service mints a handle for the URL of the content. At a technical level this would involve an HTTP POST request to the **mint** service, for example:

`https://services.ands.org.au:8443/pids/mint?type=URL&value=http://cms.org1.au/content1location`

Since **mint** is an administrative service, the XML fragment containing client-specific metadata must also be included in the request body:

```

<request name="mint">
  <properties>
    <property name="appld" value="<ANDS-issued identifier>"/>
    <property name="identifier" value="MyCMS"/>
    <property name="authDomain" value="ldap.mycms.org.au"/>
  </properties>
</request>

```

The `mint` response would contain the following fragment in the response body, any or all of which may be used by the CMS for auditing or metadata.

```

<?xml version="1.0" encoding="UTF-8"?>
<response type="success">
  <identifier handle="10378.2/99">
    <property index="1" type="URL" value="http://cms.org1.au/content1location"/>
  </identifier>
  <timestamp>2009-05-15T01:41:11Z</timestamp>
  <message type="user">Successfully authenticated and created handle</message>
</response>

```

In this scenario it would be expected the CMS would store the handle to ensure it can manage the URL associated with it into the future.

When content location changes within the CMS domain either by a user re-organising their content or through CMS maintenance the CMS would issue a `modifyValueByIndex` request using index value 1 (indexes are currently sequential starting at 1 and in this scenario we have only one value associated with a handle). Again an HTTP POST is sent to the following URL:

<https://services.ands.org.au:8443/pids/modifyValueByIndex?handle=10378.2/99&value=http://cms.org1.au/newcontent1location&index=1>

Since `modifyValueByIndex` is an administrative service, the XML fragment containing client-specific metadata must also be included in the request body:

```

<request name="mint">
  <properties>
    <property name="appld" value="<ANDS-issued identifier>"/>
    <property name="identifier" value="MyCMS"/>
    <property name="authDomain" value="ldap.mycms.org.au"/>
  </properties>
</request>

```

The `modifyValueByIndex` response would contain the following fragment in the response body, any or all of which may be used by the CMS for auditing or metadata.

```

<?xml version="1.0" encoding="UTF-8"?>
<response type="success">
  <identifier handle="10378.2/99">
    <property index="1" type="URL" value="http://cms.org1.au/newcontent1location"/>
  </identifier>
  <timestamp>2009-05-15T02:41:11Z</timestamp>
  <message type="user">Successfully modified handle value</message>
</response>

```

It is the responsibility of the CMS to ensure all URLs associated with the content's handles

are kept up to date in the Persistent Identifier service.