Faster Access to CILogon’s Persistent Store

Motivation

Current access is through the PHP layer which in turn makes a call to a Perl database application. This is causing some very noticeable performance issues. The main reasons are

- The PHP library creates a completely new Perl interpreter for each call. This has a fixed overhead of .2 sec
- The Perl application requires roughly .5 seconds to create a connection to the database.

No connection pooling is possible because of issue #1, so regardless of the database’s speed, there is a fixed .7 second delay for *each* call to the database. The net effect is an unacceptably slow web application.

Proposed solution

The Java delegation application manages all of its state with connection pooling and caches. Although there is a small initial cost for initialization whenever Tomcat is restarted (< 3 sec.) , access afterwards runs consistently in 20ms - 30 ms range. This is an improvement over the current method by an easy factor of 10. It makes sense to expose this to the rest of the CILogon Service itself. Under Tomcat, each web application with its various servlets share a common instance. Rather than set up a separate dedicated web application to serve this, it will be a simple servlet running under the delegation application. The main reason for this is that otherwise there will be both threading and latency issues involved in updating and accessing information which would be very difficult to resolve.

Security and access

For performance reasons, access will be via http (no SSL) to Tomcat directly on port 8080 (avoiding the translation from Apache through the AJP connector). Since this in effect would make *everything* in the database world readable, only localhost access will be permitted.
The path to the database servlet will be **http://localhost:8080/delegation/dbService**. This is the standard and the strongly suggested way to do this.

Alternately, If you are running it in another environment, you should enable SSL and may set DB Service users in the configuration file.

```
<dbService enabled=”true”>
  <user>
     <user name=”…” hash=”...”/>
     <!-- more users →
  </users>
</dbService>
```

In this case, the user has a name and an issued password. The SHA-1 hash of the password is recorded here. When calling the DB Service, the user name and password must be supplied with each call. the default is to have this disabled (so if this section is missing from your configuration file it is disabled) since the suggested way to do this is by not having a publicly accessible Tomcat.

The API itself

All access will be via HTTP Get. Technically we should only allow HTTP Post on requests which may result in changes to the server (e.g. a creating a user) and HTTP Get for information requests, however, this would make it awkward to use. The aim is to make it simple enough that clients can hand-code the request and parse the result without having to resort to a library or other mechanism to do so.

Tomcat Advisory

(Note applies to Tomcat 5 only.) Allowing access via HTTP GET does have a potential limit.

Tomcat will refuse to accept any request over a default size of 8192 bytes, so in particular setting IDPs might fail at some point. The fix is to set the maxHttpHeaderSize parameter in the connector *e.g.*, as follows

```
 <Connector port="8080"
   protocol="HTTP/1.1"
   maxHttpHeaderSize="65535"
   connectionTimeout="20000"
   redirectPort="8443" />
```

Failure to do so will lead, at some point to (misleading) HTTP 404 errors.

Making a request

Every request will go to the same address:

http://localhost:8080/delegation/dbService.

Requests are standard HTTP GET with key=value pairs. One of these is required, “action=XXXX”, where XXXX determines what is to be done by the server. The remaining key/value pairs are parameters for the call. All arguments in a post or get will be URL encoded but key/value pairs may be in any order. Required arguments must be present or an error will be returned. Generally (except for a list of IDPs) repeated arguments will cause a duplicate argument message to be generated.

Format of a response from the server

The general serialized form for an object (which is always the body of the response and is urlencoded) is:

```
  status=XXX
  key1=value1
  key2=value2
  …
```

where keys may be repeated for multi-valued objects. Each key/value pair is followed by a linefeed, except the final one. If a value is missing, it will still be present in the form “key=” (no value. Each section below detailing an API call will list what are and are not acceptable values. The status line is always present. The next section details the possible values.

Status Codes and Error Conditions

It may occur that there are errors during the processing of a request, e.g. if the parameters are incorrect. Rather than confuse server errors with data errors, we follow the following policy: Only actual errors with the servlet itself will result in an HTTP status code different than 200. E.g. a 404 Not Found error can only have come from the web server itself and means there is a problem with the request rather than meaning, say, that a user was not found. The first line of each response (and the only required line) is the status. This will be recorded in the body of the response according to the following table. All operations except saving the list of IDPs can return a duplicate parameter exception. It is not the task of this servlet to disambiguate or merge conflicting requests. Duplicate arguments are in general not allowed for any argument, including optional ones. The exception to this rule is setting a list of IDPs. All operations can return a missing parameter error for a required parameter.

Value	Decimal	Hex	Comment
OK	0	0x0	Normal return
ActionNotFound	1	0x1	No such action is supported by this service. Normally this indicates that the action value was mis-typed.
NewUser	2	0x2	A new user was created
UserUpdated	4	0x4	An existing user was updated
UserNotFound	6	0x6	The requested user was not found. Returned by hasUser, getLastArchivedUser, removeUser.
UserExists	8	0x8	The requested user exists. Returned by hasUser.
UserExistsError	1048481	0xFFFA1	An attempt to get a nonexistent user failed.
UserNotFoundError	1048483	0xFFFA3	An attempt to update or otherwise access a nonexistent user failed.
TransactionNotFound	1048485	0xFFFA5	No such transaction for the given temporary credential was found
IDPSaveFailed	1048487	0xFFFA7	Saving the list of idps failed.
DuplicateParameterFound	1048561	0xFFFF1	A duplicate argument was supplied.
InternalError	1048563	0xFFFF3	Some error internal to the server occurred during processing. Consult the server logs.
SaveIDPFailed	1048565	0xFFFF5	There was a problem saving the list of IDPs.
MalformedInputError	1048567	0xFFFF7	An input was of the incorrect format. E.g. an illegal uri or a string that cannot be parsed into an integer.
MissingParameterError	1048569	0xFFFF9	A required argument was not found.
NoRemoteUser	1048571	0xFFFFB	Calls that require the remote user will fail with this message if it is not supplied.
NoIdentityProvider	1048573	0xFFFFD	Calls that require the identity provider will

			fail with this message if it is not supplied.
Transaction not found	65537	10001	No transaction with the given identifier could be found
Expired token	65539	10003	The token for this request has expired.
Create transaction failed	65541	10005	General exception when a transaction cannot be created
Unknown callback	65543	10007	The supplied callback id not in the list of registered callbacks
Client id missing	65545	10009	No client identifier has been supplied with this request
No registered callbacks	65547	1000A	The client has no callbacks registered. (Normally this implies an incomplete registration)
Unknown client	65549	1000C	The identifier does not match any client
Unapproved client	65551	1000E	This client has been registered but has not yet been approved.

Date formatting

As with the rest of the system, all dates are ISO 8601 compliant of the form

```
   yyyy-mm-ddThh:mm:ss.xxxxZ
```

Where the infixed T prefixes the time and the final Z indicates this is GMT. E.g.,

```
   2010-11-17T17:09:19.692Z
```

Dates are needed for instance on user and archived user objects.

The API Calls

GetUserID(remoteUser | EPPN | EPTID | OpenID|oidc, idp)

What's it do: Given the at least one of the identifiers and the idp, this will return the internal unique identifier for this user.

Examples:

http://localhost:8080/delegation/dbService?action=getUserID&remote_user=bob
%40foo.edu&idp=urn%3Amace%3Aincommon%3Auiuc.edu

http://localhost:8080/delegation/dbService?action=getUserID&eppn=bob
%40foo.edu&eptid=https%3A%2F%2Fidp.uni.edu.au%2Fidp%2Fshibboleth!https%3A%2F

%2Fmanager.aaf.edu%2Fshibboleth!Mza74xVcOOJ%2FI%2FZ3NFFY86%2BnfOk&idp=urn%3Amace%3Aincommon%3Aaaf.edu

http://localhost:8080/delegation/dbService?action=getUserID&open_id=bob2468%40myopen.com&idp=urn%3Amace%3Aincommon%3Amyopen.com

Request key/value pairs.

Key	Value	Comment
action	getUserID	Required. Get the unique identifier for the user given the remote user and idp
remote_user	remote user	Required*
eppn	EduPerson Principal Name	Required*
eptid	EduPerson Targeted ID	Required*
open_id	Open ID	Required*
oidc	Open ID Connect identifier	Required*
idp	the identity provider	Required.

* = at least one of the values must be present.

Response key/values pairs

Key	Value	Comment
status	OK, UserNotFound, ActionNotFound	
user_uid	the unique identifier	

A complete response, for example:

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: application/x-www-form-urlencoded
Transfer-Encoding: chunked
Date: Fri, 19 Nov 2010 22:51:55 GMT

48
status=OK
user_uid=https%3A%2F%2Fcilogon.org%2FserverA%2Fusers%2F1803
```

GetUser

What's it do: This set of calls will always return the complete set of user information as stored. Missing values will be returned as either "key=" with no value or omitted completely
Note: This has 2 different possible polymorphic calls associated. We will give each in terms of its arguments separately

Case #1 - getUser(uid)

Example:
Request:

```
http://localhost:8080/delegation/dbService?action=getUser&user_uid=http%3A%2F
%2Fcilogon.org%2FserverA%2Fusers%2F119
```

Body of response:

```
status=OK
remote_user=gaynor%40illinois.edu
idp=urn%3Amace%3Aincommon%3Auiuc.edu
idp_display_name=University+of+Illinois+at+Urbana-Champaign
first_name=Jeffrey
last_name=Gaynor
user_uid=http%3A%2F%2Fcilogon.org%2FserverA%2Fusers%2F119
email=gaynor%40illinois.edu
serial_string=A119
distinguished_name=%2FDC%3Dorg%2FDC%3Dcilogon%2FC%3DUS%2FO%3DUniversity+of+Illinois+
     at+Urbana-Champaign%2FCN%3DJeffrey+Gaynor+A119+email%3Dgaynor%40illinois.edu
create_time=2010-05-14T22%3A23%3A23.397Z
two_factor=sjkhsdkkfhsirandomStringishThingish
```

Request key/value pairs.

Key	Value	Comment
action	getUser	Required.
user_uid	the unique id of the user	Required.

Response key/value pairs

Key	Value	Comment
status	OK, ActionNotFound UserNotFoundError, MissingParameter, DuplicateParameter	
user_uid		Identical to the argument
remote_user		

idp		
idp_display_name		
first_name		
last_name		
email		
serial_string	The serial string (e.g. A123) as computed by the server	
distinguished_name	The DN as computed by the server	
create_time		ISO 8601 formatted date field.
two_factor		A string with two factor information if available. Omitted if none is found.

Case #2 - getUser(remoteUser|EPPN|EPTID|OpenID|oicd, idp,...)

Note: This version of getUser is actually a shortcut for one of

- creating a new user,
- fetching a user with given identifier and idp fields
- checking such an existing user, updating its fields and returning the result.

The identifier and idp are the only required arguments. Omitting the others will set them to being empty, so all 6 parameters are always required. This means in particular this should be viewed as an update to a user which happens to return the user too. If the user has changed, then it is automatically archived, the most recent of which may be recovered with the getLastArchivedUser call detailed below.

Request key/value pairs

Key	Value	Comment
action	getUser	Required.
remote_user		Required*
eppn		Required*
eptid		Required*
open_id		Required*
oidc		Required*

idp		Required.
idp_display_name		
first_name		
last_name		
email		

* = at least one of these must be included.

Response key-value pairs

Key	Value	Comment
status	OK,ActionNotFound, UserNotFound, MissingParameter, DuplicateParameter	
user_uid		
remote_user		Identical to the argument*
eppn		Identical to the argument*
eptid		Identical to the argument*
open_id		Identical to the argument*
oidc		identical to the argument*
idp		Identical to the argument
idp_display_name		Identical to the argument
first_name		Identical to the argument
last_name		Identical to the argument
email		Identical to the argument
serial_string	See above	
dn	See above	

* = **IF** this is supplied as an argument, it will be identical. It is possible that other fields may be returned as well. E.g. If both eppn and eptid are stored, you may request a user by their eptid only. Then you would get that plus the stored eppn.

two_factor		Two factor information, if any is found. Omitted otherwise.

RemoveUser(uid)

What's it do: Removes the user from the store, archiving it beforehand.
Example:

Request: http://localhost:8080/delegation/dbService?action=removeUser&user_uid=https%3a%2f%2fcilogon.org%2fserverA%2fusers%2f2133

Body of response:

```
status=0
```

Request key/value pairs

Key	Value	Comment
action	removeUser	Required.
user_uid	The user's unique identifier	Required

Response key/value pairs

Key	Value	Comment
status	OK, ActionNotFound, MissingParameter, DuplicateParameter, UserNotFoundError	If the asserted user is not in the store, an error is returned.

Set all IDPs

What's it do: Save the entire given list. This replaces the current list of IDPs.
Note: This will not save an empty list of IDPs. If there is an error, this will try to rollback the save.
Example: (With url encoding)

```
http://localhost:8080/delegation/dbService?action=setAllIdps&idp_uid=urn%253Aidentity
%252Fprov%252F1290201592400&idp_uid=urn%253Aidentity%252Fprov%252F1290201595564
```

Request key/value pairs

Key	Value	Comment
action	setAllIdps	Required.
idp_uid	A url encoded identity provider	Required (multiples allowed)

Response key/value pairs

Key	Value	Comment
status	OK, ActionNotFound, MissingParameter	A missing parameter exception occurs if no idps are supplied.

Get All IDPS

What's it do: Get the list of all current IDPs
Example:
Request:`http://localhost:8080/dbService?action=getAllIdps`

Body of Response:

```
status=OK
idp_uid=urn%3Amace%3Aincommon%3Auchicago.edu
idp_uid=urn%3Amace%3Aincommon%3Aidp.protectnetwork.org
idp_uid=urn%3Amace%3Aincommon%3Albl.gov
```

Request key/value pairs

Key	Value	Comment
action	getAllIdps	Required.

Response key/value pairs

Key	Value	Comment
status	OK, ActionNotFound	
idp_uid	a url encoded identity provider	There will be many of these normally, one per line.

Get Portal Parameters

What's it do: Gets the portal's parameters (callback, name, etc.) based on the temporary credential.
Request key/value pairs

Key	Value	Comment
action	getPortalParameter	Required.
oauth_token	The temporary credential	Required.

Response key/value pairs

Key	Value	Comment
status	OK, ActionNotFound, TransactionNotFound, MissingParameter, DuplicateParameter	
oauth_token		same as argument
cilogon_callback	The callback uri	
cilogon_success	The success uri	
cilogon_failure	The failure uri	
cilogon_portal_name	The name of the portal	

GetLastArchivedUser

What's it do: This will take the id of a user and return the last user archived under that id, if there is one.
Note: The internal information about the archive (the date at which the user was archived) is not returned. Just a user object. Also, users are archived as a matter of course when during the getUser call, if the user's information has changed. See the information under that entry.

Request key/value pairs

Key	Value	Comment
action	getLastArchivedUser	Required.
user_uid	The user's unique identifier	Required

Response key/value pairs

Key	Value	Comment
status	OK, DuplicateParameter, ActionNotFound, UserNotFoundError	
user_uid		
remote_user		
eppn		
eptid		

open_id		
oidc		
idp		
idp_display_name		
first_name		
last_name		
email		
serial_string		
dn		

setTwoFactorInfo(userid, twoFactorInfo)

What's it do? Set the two factor information for the given user id.
Note: The two factor information is simply an opaque string. No parsing or other processing of it will be done.

Request key/value pairs

Key	Value	Comment
action	setTwoFactorInfo	
user_uid	user identifier	
two_factor	String	Opaque string. No processing of any sort done. This is not assumed to be binary.

Response key/value pairs

Key	Value	Comment
status	ok, user not found	

getTwoFactorInfo(userid)

What's it do? This will retrieve any two factor information from the server for the given user id.

Request key/value pairs

Key	Value	Comment
action	getTwoFactorInfo	
user_uid	user unique identifier	

Response key/value pairs

Key	Value	Comment
status	ok, user not found	
user_uid	user id	Same as in the request.
two_factor	string	as per setter, this is assumed to be an opaque string. Omitted if no user information found

setTransactionState()

What's it do? (OAuth2) This will retrieve the transaction corresponding to a given authorization grant.

createTransaction

What's it do? (OAuth2) This creates a transaction. Note that the way CILogon works, the call goes to a PHP layer first which in turn has to do various bits of setup. This call is made by the PHP layer to set up an OAuth transaction and is the first call of any sort after a request is received.

getClient(client_id)

What's it do? (OAuth2) This will retrieve the information for a given client.