CILogon DB Service

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**.

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.

Basic Operation

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.

Note: There is an appendix at the end of this document with various tables sorting these for quick reference.

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.
IDPChanged	10	0xA	User was found but the IDP changed.
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

checkUserCode(user_code)

What's it do: This will take the user_code and verify that it is currently active.

Request key/value pairs.

Key	Value	Comment
action	checkUserCode	Required
user_code		Required

Response key/value pairs.

Key	Value	Comment
status	Ok missing parameter service unavailable transaction not found expired token	The service unavailable response is if the service does not have the device flow enabled. Expired token refers to the auth grant for the transaction.
client_id	The client id	
grant	The id for the transaction	This is base 32 encoded
scope	Scopes, if any, in the initial request	This is a blank delimited list
user_code	The user_code passed in	This helps the requester identify this response

See also: userCodeApproved

createTransaction

What's it do: This creates the transaction after authorization. This returns the code and other information.

Request key/value pairs.

Key	Value	Comment
action	createTransaction	Required
client_id	The client identifier	
scopes	The scopes passed in the initial request.	
state	Opaque string	

Response key/value pairs.

Key	Value	Comment
status	Ok missing argument missing client id no scopes malformed scope malformed input unknown client unapproved client create transaction failed internal error	
code	The authorization grant	This is base 32 encoded
scope	JSON array	This is the set of scopes that the client is actually allowed. It may not be the same as what was passed in.
state	Echos back passed in state	This is so clients can set a value to track transactions

See also: setPortalParameters, setTransactionState, setTwoFactorInfo, getTwoFactorInfo

getAllIDPS

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.

See also: setAllIDPs

getClient(client_id)

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

Request key/value pairs

Key	Value	Comment
action	getClient	Required
client_id	The unique client identifier	

Response key/value pairs

Key	Value	Comment
status	ok, missing client id, client not found	
client_callback_uris	Callback uris	List is blank delimited.
client_creation_timestamp	ISO 8601 date	
client_email		
client_home_uri		
client_id		
client_limited_proxies		
client_name		
client_refresh_lifetime	lifetime in milliseconds	

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 1: 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.

Note 2: Archiving users was necessary for tracking revisions to serial strings – a requirement for X509 certificates. As of May 2025, CILogon will no longer issue those and this call will be official deprecated. It may still be called by some legacy code, but it should not be used in new code.

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
dn		
email		
eppn		
eptid		
first_name		
idp		
idp_display_name		
last_name		
oidc		
open_id		
remote_user		
serial_string		
status	OK, DuplicateParameter, ActionNotFound, UserNotFoundError	
user_uid		

getPortalParameters

What's it do: Gets a partial transaction (callback, name, etc.) based on the authorization grant (aka the temporary credential). This is intended for getting basic information when the user is in the process of logging in, not the entire transaction, which could be quite massive!

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	
cilogon_callback	The callback uri	
cilogon_failure	The failure uri	
cilogon_portal_name	The name of the portal	
cilogon_success	The success uri	
oauth_token		same as argument

See also: createTransaction, setTransactionState, setTwoFactorInfo, getTwoFactorInfo

getTwoFactorInfo(user_uid)

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	
two_factor	string	as per setter, this is assumed to be an opaque string. Omitted if no user information found
user_uid	user id	Same as in the request.

See also: setTwoFactorState

getUser(user_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	

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

See also: getUser(remoteUser..), getUser(user_uid), getUserID, hasUser, removeUser

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.
email		
eppn		Required*
eptid		Required*
first_name		
idp		Required.

idp_display_name		
last_name		
oidc		Required*
open_id		Required*
remote_user		Required*

* = at least one of these must be included.

Response key/value pairs

Key	Value	Comment
status	OK,ActionNotFound, UserNotFound, MissingParameter, DuplicateParameter, IDPUpdated	
dn	See above	
email		Identical to the argument
eppn		Identical to the argument*
eptid		Identical to the argument*
first_name		Identical to the argument
idp		Identical to the argument
idp_display_name		Identical to the argument
last_name		Identical to the argument
oidc		identical to the argument*
open_id		Identical to the argument*
remote_user		Identical to the argument*
serial_string	See above	
two_factor		Two factor information, if any is found. Omitted otherwise.
user_uid		

* = **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.

See also: getUser(user_uid), getUserID, hasUser, removeUser

getUserID

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
eppn	EduPerson Principal Name	Required*
eptid	EduPerson Targeted ID	Required*
idp	the identity provider	Required.
oidc	Open ID Connect identifier	Required*
open_id	Open ID	Required*
remote_user	remote user	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
```

See also:getUser(user_uid), getUser(remoteUser…), getUserID, hasUser, removeUser

hasUser(user_uid)

What's it do: This will check if a user with a given uid is in the system,

Request key/value pairs

Key	Value	Comment
action	hasUser	
user_uid	The unique identifier for the user	Required

Response key/value pairs

Key	Value	Comment
status	User exists user not found missing parameter	Note that ok is not a response from this call.

See also: getUser(uiser_uid), getUser(remoteUser…), getUser(user_uid), removeUser(user_uid)

removeUser(user_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.

See also: getUser(uiser_uid), getUser(remoteUser…), getUser(user_uid), hasUser(user_uid)

setAllIDPs

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.

See also: getAllIDPS

setTransactionState

What's it do? This sets basic information in a transaction during the user's initial logon. It sets the user uid in the transaction.

Request key/value pairs

Key	Value	Comment
action	setTransactionState	Required
auth_time	The timestamp when the user uathenticated	This is in milliseconds
cilogon_info	The username MyProxy expects	When getting x509 certs only.
code	The authorization grant	This is the unique identifier for

		the transaction
loa	Level of assurance	
user_uid	Unique user id	

Response key/value pairs

Key	Value	Comment
status	Ok, missing argument, expired token transaction not found QDL error QDL runtime error	Missing argument is if the code is missing

See also: createTransaction, setPortalParameters setTwoFactorInfo, getTwoFactorInfo

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

Response key/value pairs

Key	Value	Comment
status	ok, user not found	

See also: getTwoFactorInfo

userCodeApproved(user_code)

What's it do? This is used in the device flow. It will approve the code (allowing the user to get a token). It may also unapprove a user code. Normally you check the user code before approving it.

Request key/value pairs

Key	Value	Comment
action	userCodeApproved	Required
approved	0 unapprove 1 approve (default)	Optional. If not present, approve the code.
user_code	The user code generated by the system	Required.

Reponse key/value pairs

Key	Value	Comment
status	Ok missing parameter transaction not found expired user code	

See also: checkUserCode

Appendix

The basic philosophy is that even number indicate some sort of success + information, odd numbers represent that an error has happened.

List of success codes

NAME	Hex	Decimal
STATUS_OK	0x0	0
STATUS_NEW_USER	0x2	2
STATUS_USER_SERIAL_STRING_UPDATED	0x4	4
STATUS_USER_NOT_FOUND	0x6	6
STATUS_USER_EXISTS	0x8	8
STATUS_IPD_UPDATED	0xA	10

Alphabetical table of error codes

NAME	Hex	Decimal
STATUS_ACTION_NOT_FOUND	0x1	1
STATUS_CLIENT_NOT_FOUND	0xFFFFF	1048575
STATUS_CREATE_TRANSACTION_FAILED	0x10005	65541
STATUS_DUPLICATE_ARGUMENT	0xFFFF1	1048561
STATUS_EPTID_MISMATCH	0x100001	1048577
STATUS_EXPIRED_TOKEN	0x10003	65539
STATUS_IDP_SAVE_FAILED	0xFFFA7	1048487
STATUS_INTERNAL_ERROR	0xFFFF3	1048563
STATUS_MALFORMED_INPUT	0xFFFF7	1048567
STATUS_MALFORMED_SCOPE	0x10013	65555
STATUS_MISSING_ARGUMENT	0xFFFF9	1048569
STATUS_MISSING_CLIENT_ID	0x10009	65545
STATUS_NO_IDENTITY_PROVIDER	0xFFFFD	1048573
STATUS_NO_REMOTE_USER	0xFFFFB	1048571
STATUS_NO_SCOPES	0x10011	65553
STATUS_PAIRWISE_ID_MISMATCH	0x100003	1048579
STATUS_QDL_ERROR	0x100007	1048583
STATUS_QDL_RUNTIME_ERROR	0x100009	1048585
STATUS_SAVE_IDP_FAILED	0xFFFF5	1048565

STATUS_SERVICE_UNAVAILABLE	0x10015	65557
STATUS_SUBJECT_ID_MISMATCH	0x100005	1048581
STATUS_TRANSACTION_NOT_FOUND	0x10001	65537
STATUS_TRANSACTION_NOT_FOUND	0xFFFA5	1048485
STATUS_UNAPPROVED_CLIENT	0x1000F	65551
STATUS_UNKNOWN_CLIENT	0x1000D	65549
STATUS_USER_EXISTS_ERROR	0xFFFA1	1048481
STATUS_USER_NOT_FOUND_ERROR	0xFFFA3	1048483

Error codes sorted by numeric value:

Hex	Name	Decimal
0x1	STATUS_ACTION_NOT_FOUND	1
0x10001	STATUS_TRANSACTION_NOT_FOUND	65537
0x10003	STATUS_EXPIRED_TOKEN	65539
0x10005	STATUS_CREATE_TRANSACTION_FAILED	65541
0x10009	STATUS_MISSING_CLIENT_ID	65545
0x1000D	STATUS_UNKNOWN_CLIENT	65549
0x1000F	STATUS_UNAPPROVED_CLIENT	65551
0x10011	STATUS_NO_SCOPES	65553
0x10013	STATUS_MALFORMED_SCOPE	65555
0x10015	STATUS_SERVICE_UNAVAILABLE	65557
0xFFFA1	STATUS_USER_EXISTS_ERROR	1048481
0xFFFA3	STATUS_USER_NOT_FOUND_ERROR	1048483
0xFFFA5	STATUS_TRANSACTION_NOT_FOUND	1048485
0xFFFA7	STATUS_IDP_SAVE_FAILED	1048487
0xFFFF1	STATUS_DUPLICATE_ARGUMENT	1048561
0xFFFF3	STATUS_INTERNAL_ERROR	1048563
0xFFFF5	STATUS_SAVE_IDP_FAILED	1048565
0xFFFF7	STATUS_MALFORMED_INPUT	1048567
0xFFFF9	STATUS_MISSING_ARGUMENT	1048569
0xFFFFB	STATUS_NO_REMOTE_USER	1048571
0xFFFFD	STATUS_NO_IDENTITY_PROVIDER	1048573
0xFFFFF	STATUS_CLIENT_NOT_FOUND	1048575
0x100001	STATUS_EPTID_MISMATCH	1048577
0x100003	STATUS_PAIRWISE_ID_MISMATCH	1048579
0x100005	STATUS_SUBJECT_ID_MISMATCH	1048581
0x100007	STATUS_QDL_ERROR	1048583
0x100009	STATUS_QDL_RUNTIME_ERROR	1048585