OS2 Command Line Client Manual

Version 5.1.1

Introduction

This is a fully featured command line client for OA4MP. It supports pretty much anything you need for doing flows with any Oauth 2.0 service, but is particularly aimed at OA4MP. It support the authorization code flow, the device flow and RFC7523 flows.

Getting it

The latest version of the jar is found at

https://github.com/ncsa/OA4MP/releases/latest/oa2-client.jar

and there is a script to run it too at

https://github.com/ncsa/OA4MP/releases/latest/oa2-client

Running it

Once you have the two you can either edit the script to point to wherever you installed it or can just manually issue

```
java jar $path_to_jar/oa2-client.jar
Warning: no configuration file specified. type in 'load --help' to see how to load
one.
No configuration loaded.
client>
```

and that is the command prompt that shows it is ready and waiting. This has no arguments, so you would need to **load** (see below) the configuration you want. Alternately, you can just specify these at the command line (which is really all the script does):

```
java jar $path_to_jar/oa2-client.jar -cfg path_to_config_file -name
name_of_config_in_file
client>
```

No message means everything loaded fine and you are ready to roll.

Quick reference

Base CLI commands

Each type of command line interface (CLI) in OA4MP is based on the same code, hence there are things that are common to all of them. These commands are generally prefixed with a /. General things this does

- prints help on these commands (issue a **/?** at the prompt).
- manages command history for display or repeat, loading and saving.
- lists the commands in the current components

These are all prefixed with a "/" and refer to running various parts of the CLI, such as the history of input and re-running commands. This derives from a general command line client interface used through OA4MP, so these work in other components too (such as the CLI to manage stores). When in doubt, issue **/?** to get help. Generally something useful is printed if you supply an argument of **-- help** (Note the *double* hyphen!)

General commands:

/exit /q = exit this component
/? = print help
/commands = list all of the currently available commands.
/trace on | off = turn *low level* debugging on or off. Use with care.

Command buffer

These are understood at all times and are interpreted before any commands are issued.
/c = clear the command history
/l path = load the command history saved in the path
/w path = write the command history to the given file

Command history:

/h [index] = either print the entire command history (no argument) or re-execute the command at the given index.
/r = re-evaluate the most recent (0th index) command in the history.
This is equivalent to issuing **/h 0**

General commands

These are what does the task for the CLI, such as being an OAuth client or managing a set of stores. This reference is designed to be quick. It will tell you where to look for things, but the absolute most correct manual is included with the CLI itself and online. Whenever you are not sure what to do

--help

(Note the double hyphen!) will give you something. It also works for each of the commands below. So if you were not sure about what the **load** command did, then you type in

client>load --help

and a bunch of help would issue forth.

Commands to manage state

- **load** - load a configuration, replacing the current.
- **read** - read a stored session from a given file
- **save_cert** - save any cert to a given file
- **write** - write the current state of this program to a file. This lets you resume where you were exactly. If you want to test long-term behavior of tokens, **write** the state to a file and later **read** it.

Commands to manage CLI behavior

- **echo** - echos the input to the consoles. Mostly used in batch mode so the user can see what commands are executing.
- **set_output_on** - mostly used in batch files. If set to false, then all output is discarded.
- **set_verbose_on** - sets the output to be chattier.
- **version** - the current version of this program
- **print_help** - print out help about these specific commands (environment, display)

Commands for environment

- **clear_env** - clear all environment variables
- **get_env** - print the value of a single variable
- **print_env** - print all the variables.
- **read_env** - read a file containing variables
- **save_env** - save the current environment to a file
- **set_env** - set a key/value pair

The environment is a set of key/value pairs that is used to pre-process all commands. It may be either created on the fly or loaded/saved. Any key may be referenced in any command and this is replaced by its value then executed. The environment is kind of useful at the command line and extremely so in batch files since you can use it to store sets variables and import them, leaving your batch script free of hard-coded values.

E.g.

```
client>set_env my_file “/opt/cilogon-oa2/var/temp/poloc-test.json”
client>write ${my_file}
```

Would write the current state to the given file. All of these are suffixed with **_env**.

OAuth and related commands

Parameter commands

- **clear_all_params** - clear all parameters.
- **get_param** - list the parameters for a request
- **set_param** - set the parameters for a request
- **rm_param** - remove a parameter for a request

Parameters are just that – parameters sent along additionally with requests to the service. There are 3 places that these may be sent, so there are 3 flags that determine which request these go with.

- **-a** or **-auth** is used for parameters sent in the initial (authorization) request.
- **-t** or **-token** is used for parameters sent in the token or refresh request.
- **-x** or **-exchange** is used for parameters sent as part of a token exchange

Note that whatever you specify is sent so it is always a good idea to enclose whatever you want to send in quotes.

E.g.

Here is a snippet that is used to set all three requests at once

```
set_param -a scope "read:/home/jeff x.y: write:"
set_param -t scope "read:/home/jeff x.y: write:/data/cluster"
set_param -x scope "read:/home/jeffy x.y:/abc/def/ghi write:/data/cluster1 x.z:/any"
```

Information about tokens and such

- **asset** - print the asset. This is used internally and is sometimes useful, but generally not of interest.
- **claims** - print any claims. Note that after each request, this may be updated.
- **show_raw_id_token** - show the last id token as a JWT. This is apt to be messy, but there it is.
- tokens - show all tokens, along with any information about expiration etc. available.

OAuth commands

- **clear** - clear the environment. This does not clear set parameters, there is a flag for that.
- **exchange** - (Requires: **get_at**). Exchange your access or refresh token
- **get_at** - (Requires **get_grant**) Initial get an access and (maybe) an refresh token.
- **get_cert** - (Requires: **get_at**) Get a cert. Also, your client must use the getcert scope.
- **get_grant** - (Requires: **set_uri**) Processes the callback from the browser after authorization.
- **get_rt** - (Requires: **get_at**) Refresh to token. Note this is *not* the same as the token exchange
- **get_user_info** - (Requires: **get_at**) Get whatever is at the userinfo endpoint.
- **revoke** - (Requires: **get_at**) Revoke a token
- **set_uri** - First call that generates the request URI for the service. Paste this into your browser.

Most common sequence

Here is a sample session.

```
client>load
config file = /home/ncsa/dev/csd/config/client-oa2.xml, config
name=dev:command.line
Remember that loading a configuration clears all current state, except parameters.
```

This lets us see what configuration file is in use and which configuration is currently active.

```
client>set_uri
URL copied to clipboard:
https://dev.cilogon.org/authorize?
scope=edu.uiuc.ncsa.myproxy.getcert+org.cilogon.userinfo+openid+profile+email&respo
nse_type=code&redirect_uri=https%3A%2F%2Flocalhost%3A9443%2Fclient%2Fnot-
ready&state=WmUFbohL3EnPJyZb5gn90eXPcvb2dz8oSfAaZmaf6aQ&nonce=Clbl8xO6Ey877Ce8LlOmP
pFLvLxjUewmQWOOd9gtYbY&prompt=login&client_id=dev%3Acommand.line
```

This creates the correct (reallyl messy) url. If your computer has a clipboard available, this is copied to the clipboard for you. Now you head to your browser, paste this in, hit return and you should get an error (if your client is properly configured, since OAuth is trying to do a redirect to your client which is not a browser. Now, highlight the URL in the browser and copy it to the clipboard. Issue the following to copy it (if there is no clipboard, paste it as the argument to **get_grant**).

```
client>get_grant
grant copied to clipboard.
grant=https://dev.cilogon.org/oauth2/1dd5515372b7dd81a915c2b3b73ff452?
type=authzGrant&ts=1612383342334&version=v2.0&lifetime=900000
```

Now we can get the access token:

```
client>get_at
default access token =
https://dev.cilogon.org/oauth2/4c78da1852b7920a7e775aac4c3e509c?
type=accessToken&ts=1612383366545&version=v2.0&lifetime=900000
   expires in = 900000 ms.
   valid until Wed Feb 03 14:31:06 CST 2021
default refresh token =
https://dev.cilogon.org/oauth2/20c8421b94be2db2bb06ca880c83b2cb?
type=refreshToken&ts=1612383366545&version=v2.0&lifetime=1000000000
   expires in = 1000000000 ms.
   valid until Mon Feb 15 04:02:46 CST 2021
```

If you want to see the claims that came back, issue

```
client>claims
{
  "sub": "http://cilogon.org/serverD/users/45",
  "idp_name": "Google",
  "cert_subject_dn": "/DC=org/DC=cilogon/C=US/O=Google/CN=j g D13115",
  "iss": "https://dev.cilogon.org",
  "given_name": "j",
  "nonce": "Clbl8xO6Ey877Ce8LlOmPpFLvLxjUewmQWOOd9gtYbY",
  "aud": "dev:command.line",
  "idp": "http://google.com/accounts/o8/id",
  "token_id":
"https://dev.cilogon.org/oauth2/idToken/5e713aa845e159799b7b861872199bff/
1612383353886",
  "auth_time": 1612383353,
  "exp": 1612384253,
  "iat": 1612383353,
```

```
}
```

Note that these values are specific to me and how I logged in to CILogon (a popular extension of OA4MP), so there will be quite a lot of variation in what's there. Finally, let's exchange our access token for another one.

```
client>exchange
default access token =
https://dev.cilogon.org/oauth2/4212c19f8334f43a9712489fd8cc614f?
type=accessToken&ts=1612383974420&version=v2.0&lifetime=900000
   expires in = 900000 ms.
   valid until Wed Feb 03 14:41:14 CST 2021
```

Now let's say you wanted to test that refreshes are working and want to wait quite some time. Just **write** your current state:

```
client>write -m "long term refresh token test" /home/jeff/oa4mp/long_term_test.json
```

This writes the file and puts the comment (-m flag) in it in case you ever want to read it. This is just a JSON blob. Later to get it back just fire up the client again and issue

```
client>read /home/jeff/oa4mp/long_term_test.json
loading configuration from /home/ncsa/dev/csd/config/clients.xml, named
localhost:command.line
done!
OA4MP command line client state stored on Tue Feb 09 08:53:38 CST 2021
long term refresh token test
```

Note that the message you stored is printed. It also remembered what the last configuration was and restored that.

Running this in batch mode

You may also run this by specifying an input file (commands, and lines that start with a # are treated as comments) as well as input files. These are read in sequence.