Server Scripts

Introduction

Any system that uses QDL as a scripting environment should use the format here to inject configurations. QDL will take these (JSON) and translate them into viable QDL scripts, setting up any environment, passing arguments (suitably processed) etc.

How scripts are accessed

QDL can be used to extend other systems so that QDL scripting is availabe. *E.g.*, in OA4MP, scripts are embedded in token handlers. See the token handler documentation for more. What is described here is the basic mechanism that can be utilized by any extension that wants to include QDL as its scripting language.

The reason is that many people who will want to use this are not proficient in QDL. This lets them call a script (from a library, *e.g.*) and send along a standard JSON object with no knowledge of how it works. This is probably the most common use case for scripting and lets an administrator delegate the scripting work to others without having to require them to learn yet another language. If they know the name of the script and the inputs, they can quiddle.

Configuration Format

This is the format for the configuration entry used by QDL . This is an element in JSON and may be put any place. This is the grammar for scripts:

```
{"qdl" :
   BLOCK | [BLOCK+], // Either a block or array of blocks
   }
```

```
BLOCK :
{
  CODE
   [, ARGS]
}

CODE:
  ("load" : LINE) | ("code"  : LINE | LINES)

// NOTE: "load" implies zero or more arguments. "code" has no arguments and
//       any will be ignored. It is possible to send enormous blocks of code this
//       way, but is discouraged. Put it in a script and call that.


ARGS:
  "args" : ARG | [ARG+] // Either a single arg or an array of them
```

```
ARG:
   STRING | INTEGER | DECIMAL | BOOLEAN | JSON

LINE:
   STRING

LINES:
   [LINE+]
```

Nota Bene: This is not exclusive. A server may add any other blocks of state (such as OA4MP's **xmd** for e***x***ecution ***m***eta***d***ata that the system needs to control when it executes the anaphor.

Examples

Running scripts

Loading a simple script that has no arguments

```
{"qdl":{"load":"x.qdl"}}
```

Note that if there were arguments, they would be included in the arg_list. While arguments to the script are optional (at least as far as the handler goes), some execution phase is always required so the handler knows when to run it. If you omit the execution phase, your code will never run.

Loading a script and passing it a list of arguments.

```
{"qdl":
   {
     "load":"y.qdl",
     "args":[4,true,{"server":"localhost","port":443}]
   }
}
```

This would create and run script like (spaces added)

```
script_load('y.qdl',
     4,true,from_json('{"server":"localhost","port":443"}')
   );
```

Note that the arguments in the configuration file (which is JSON/HOCON) are respectively an integer, a boolean and a JSON object. These are faithfully converted to number, boolean and stem in the arguments the script gets.

Loading a script with a single argument

```
{"qdl": {
  "load": "y.qdl",
  "args": {"port": 9443,"verbose": true,"x0": -47.5,"ssl":[3.5,true]},
 }
}
```

In this case, a script is loaded and a single argument is passed. This is converted to

```
script_load('y.qdl',
   from_json('{"port":9443,"verbose":true,"x0":-47.5,"ssl":[3.5,true]}')
  );
```

Running code directly

Running a single line of code.

Note that all examples are taken from OA4MP, which adds an **`xmd`** block for its own state management.

```
{
  "qdl": {
    "code": "claims.foo:='arf';",
    "xmd": {"exec_phase": ["post_token"]}
  }
}
```

This will assert a single claim of foo.

Running multiple lines of code.

```
{
  "qdl": {
    "code": [
      "x:=to_uri(claims.uid).path;",
      "claims.my_id:=x-'/server'-'/users/';"
    ],
    "xmd": {"exec_phase": "pre_token"}
  }
}
```

Loading multiple scripts

You may certainly mix and match anaphors, These will be passed as an array and will be run sequentially. They are simply passed as an array of scripts:

```
{
  "qdl": [
    {
      "load": "ga4gh/at.qdl",
      "args": [8443,true]
    },
    {
      "load": "ga4gh/ga4gh.qdl",
      "args": [443,"ssl_on","post_user_info"]
    }
  ]
}
```

However, there is a strong argument for not doing this, but simply having a single QDL script that organizes them. Run that script.

In this case, two scripts are run in sequence. The first is **`at.qdl`** , and the second, **`ga4gh.qdl`**. Since the state is the same between the first can set it up for the second, e.g.