QDL Scripting

Version 1.4

Introduction

Scripting – the execution of a set of commands that do not require compilation – is included in QDL. It has many feature to help with this. The basics are that any set of QDL commands in a file (extension is .qdl) can be run using your current workspace settings, just as if you were typing it in.

The basics

The requirements are

- install QDL. It is assumed installed in $QDL_HOME.
- set your preferred configuration in the $QDL_HOME/bin/qdl-run script.
- Probably add $QDL_HOME/bin to your $PATH so you can just type qdl or qdl-run as needed.

Once you have these, a good test is to execute your first script:

```
qdl-run $QDL_HOME/examples/hello_world.qdl
Hello World!
```

Which runs the basic hello world script.

If you want to play with the examples more easily, you should probably set *script_path* in the configuration (see the document in the docs directory or look at the online version). Then all you need to do is enter the name of the script to run it within the workspace. Note the you will need to pass in the correct file name to qdl-run since the operating system handles that and is unaware of things like virtual file system in QDL.

Running scripts at the command line

Running vs. loading scripts in QDL

We us two similar terms, *run* and *load*. The difference are

- *run* implies that any state/variables created in the script remain in the script. The calling environment is unknown to the script, thought eh script may return a result.
- *load* executes the contents of the script in the current environment. This is useful for,*e.g.*, initializing a bunch of variables, or loading things into the current environment. At the end of the script, everything it did is now available. Initialization scripts and boot scripts are generally loaded.

- If you are writing a script and need to have a local set of variables (frequently a good idea) put them in a

```
block[
 // statements. Any new variables created here vanish outside of the block
];
```

Mostly you want to run scripts rather than load them. It is always better to keep your environments separate unless you know exactly what the script does. An unknown script may reset or overwrite your current environment with no warning if loaded.

From the command line.

QDL as a general purpose scripting language

You can run QDL scripts at the command line like any other scripting language (e.g. python or Ruby) if you start the file with the following *shebang directive*:

```
#! /usr/bin/env -S qdl-run
```

Note that #! is available only on Unix and is processed by the operating system. If you are not using unix of some sort, see the next section for another way to run scripts. QDL ignores shebang lines (Effectively to QDL, any script with a line line that starts with #! is just a comment and is stripped out at processing.) What this does is tell the OS to look up qdl-run in your PATH (make sure this is set!) and invoke it, passing the current script. The -S tells it to pass along arguments. (Note: Some versions of unix, such as CentOS, have a slightly different syntax for env, so consult your local documentation.)

Turning your QDL script into a shell script

1. Add the above shebang line as the very first one in the file, starting in column 0.
2. Set the execute mode. i.e., issue

   chmod 751 *script*
3. Invoke it.

Any easy test if you have set things up right is in the distro $QDL_HOME/examples/hello_world.qdl should run directly from the command line as

```
bash$cd $QDL_HOME/examples
bash$./hello_world.qdl
Hello world!
```

And the source code *in toto* is

```
#! /usr/bin/env -S qdl-run
say('Hello world!');
```

If that works, you can now quiddle from the command line with wild abandon. There is another small script for testing passing arguments in the examples directory of the standard distribution called **`echo-it.qdl`**. That will just echo back whatever arguments you pass in. If you change tot he directory and try .e.g

```
bash$./echo_it.qdl foo "bar baz"
you entered the following arguments:
# 0:foo
# 1:bar baz
```

Note that arguments with embedded blanks should be enclosed in double quotes.

Running scripts directly with qdl-run

The above works fine on Unix, but not other platforms, so the generic way to run scripts is by invoking qdl-run.

```
qdl-run script_name arg0 arg1 arg2,…
```

Note: It is often a good idea to enclose arguments in double quotes and in point of fact, if there are embedded blanks it is mandatory. These may then be accessed inside the script using the script_args() function. Note that these are all strings since that is the only option supported. (If you object, please lobby to have your OS rewritten to be QDL aware.) the script_args() call is documented more fully in the documentation (included in $QDL_HOME/docs/qdl_reference.pdf) but the basics are

- script_args() – no arguments returns how many there are, e.g. 3
- script_args(n) – n is an integer in the range 0 < n < script_args() will return the n-th argument.
- script_args(-1) – a list of all arguments.

Example

```
qdl-run script_name fee fi fo fum
```

would have the following

script_args() returns 4

script_args(0) return the string ‘fee’

script_args(3) returns ‘fum’

script_args(-1) returns [fee, fi, fo, fum], *i.e.* all the args as a list.

Calling scripts in QDL: passing and using arguments

To run a script from another script, use the script_run() or script_load() call, *e.g.*,

```
script_run(‘path/to/script.qdl’ (, arg)*);
```

So the script at the given path would be invoked. The arguments may be anything – unlike invocation from a shell script, where you may only pass strings.

```
script_run('script_name', 'fee',-3.17,'fo','fum');
```

So in the script, **`script_args(0) == 'fee'`**, **`scripts_args(1) == -3.17`**, etc.

Getting input from users

QDL has a function scan:

```
scan([prompt])
```

This will print *prompt* if present and return a string of whatever the user types in. E.g.

```
    say(scan('type something>'));
type something>mairzy doats and dozey doats.
mairzy doats and dozey doats.
```

So if your script requires some user input, you may use this. There is a script in the $QDL_HOME/examples directory named **`scan_it.qdl`** with a good example.

Returning values

Scripts may return values using the return() call. There are a few items to note.

External facing scripts

If this is to be a script that is outward facing (so consumed by another non-QDL process), any result returned will be sent to standard out plus a line feed (so it will display properly at the console). Standard out best understands strings, so we use them. While you can return your massive stem, it will get converted to a string. This is a limitation of underlying OS's. In this case, it is not a bad idea at all to convert it to input form or perhaps JSON, since that is one reason JSON exists. Not having a return statement or calling `return()` (no argument) does not return a result to standard out.

```
bash$ qdl-run $QDL_HOME/examples/my_sqrt.qdl 5
2.23606797749979
bash$
```

Inward facing scripts

If the script is called by other scripts, you can return anything and that will be the output value of the script, *e.g.* if you have a script that computes approximations of a square root for an argument (this is included in the examples directory – the *script_path* in the configuration has been set to look there):

```
 script_run('my_sqrt.qdl', 5)^2
4.9999999999999968
```

Calling other scripts

Once a script is running in QDL, it has full access to the resources of the workspace. So that means that it can call other scripts that are, e.g., in a VFS. A common pattern is to have a boot script that simply starts QDL and loads other scripts to do the work.

Example

You have a script that resides in a VFS at `vfs#/bsu/scripts/init.qdl`. You need to pass it several arguments that only are external. How to run it from the command line? Write a script like bsu-init.qdl:

```
args. := script_args(-1);
script_run('vfs#/bsu/scripts/init.qdl', args.0, args.1, ...);
```

Scripts may call other scripts and there is no limit placed. Much like recursive functions, they may call themselves and this is not flagged as an error. However, recursion is generally a terrible idea if it happens without being an explicit design decision.

Running QDL from standard in, out

You may simply invoke QDL and pipe in commands as if you were sitting at the console. QDL will exit when it hits the end of the stream though.

```
qdl < my_stuff.qdl
```

will execute every command in my_stuff.qdl as if you were typing it in, line by line, including sending the output to the console. If you want to capture the output in a file, use the standard out redirect >

```
qdl < my_stuff.qdl > ~/temp/out.txt
```

would pipe `my_stuff.qdl` in and all the output would end up in `~/temp/out.txt.`

FYI

```
qdl < my_stuff.qdl >> ~/temp/out.txt
```

would append the output to the given file.