uQueryDB: Querying the MOOSDB from the Command Line

June 2018

Michael Benjamin, mikerb@mit.edu
Department of Mechanical Engineering
MIT, Cambridge MA 02139

`project-pavlab/appdocs/app_uquerydb`

1 Overview

The `uQueryDB` utility is a command-line tool for querying a MOOSDB with a logic condition provided on the command line. It finds the MOOSDB via a mission file provided on the command line, or the IP address and port number given on the command line. It will connect to the DB, register for the variables involved in the logic condition and determine if the condition holds. It will then exit with 0 if it holds or 1 otherwise. It will return its value as soon as the app has received mail for all variables involved in the logic condition. Otherwise it will wait for 10 seconds. This can be changed with the `--wait=N` parameter. If a variable in the logic condition is unknown to the MOOSDB, then the whole condition will fail after the wait period.

The motivation for `uQueryDB` is to facilitate automated testing of missions launched from within a shell script, by checking for termination criteria and automatically halting the test mission.

1.1 A Simple Example

As a simple example, try launching the alpha mission in the `s1_alpha` example mission folder. From the command line, launch with:

```
$ cd moos-ivp/ivp/missions/s1_alpha
$ ./launch.sh
```

Once this simulation has started, deploy the vehicle with the button on the viewer. At this point we can try out the `uQueryDB` utility. In another terminal window try typing:

```
$ cd moos-ivp/ivp/missions/s1_alpha
$ uQueryDB alpha.moos --condition="DB_UPTIME<130"
```

This will check the value of the MOOS variable `DB_UPTIME` which is the number of seconds that the MOOSDB has been up. The above check will simply report success until after 130 seconds has passed. The report comes in human-readable form in the terminal output:

```
$ uQueryDB alpha.moos --condition="DB_UPTIME<130"
Mission File was provided: alpha.moos
Condition: [DB_UPTIME<130]

-------------- moos connect ----------------------
  contacting a MOOS server localhost:9000 -  try 00001
  Handshaking as uQueryDB               [ok]
  DB reports async support is           [on]
  DB is running on                      leonardo
  Timing skew estimation is             [off] (not needed)
--------------------------------------------------

uQueryDB is Running:
 |-Baseline AppTick   @ 5.0 Hz
 |--Comms is Full Duplex and Asynchronous
 -Iterate Mode 0 :
   |-Regular iterate and message delivery at 5 Hz

==============================================================
uQueryDB alpha                                          0/0(1)
==============================================================
Config:
  m_sServerHost: localhost
  m_lServErport: 9005
  Max Time:      n/a
State:
  Start Time:    1640112857.63442
  Curr Time:     1640112857.83831
  Elapsed Time:  n/a
  Exit Value:    1
Config (pass/fail):
  pass_conditions: 1
  fail_conditions: 0
Result: fail: DB_UPTIME<13


InfoBuffer: (pass condition vars)
============================================
DB_UPTIME  107.35
    WFLAG  waypoints=3

InfoBuffer: (fail condition vars)
============================================
DB_UPTIME  107.35
    WFLAG  waypoints=3

InfoBuffer: (check vars)
============================================
WFLAG  waypoints=3
```

Note in the last two lines the current value of the MOOS variable DB_UPTIME is reported and the result of the logic condition is declared along with the value returned by the command, `exit(1)`.

2 Using the Output Generated by uQueryDB

In addition to the human-readable output above, the uQueryDB utility also generates a return value available on the command line. In line with common GNU/Linux tradition, this utility returns 0 when it is successful and a non-zero value when not successful. In our case uQueryDB returns 0 when the logic condition succeeds, and 1 when it does not. As a simple exercise, launch the alpha mission again as in the previous example. After it has again launched, enter the following in another terminal window:

```
$ uQueryDB alpha.moos --condition="DB_UPTIME<130" >& /dev/null
```

By redirecting to `/dev/null` the normal terminal output is suppressed but the result is generated nevertheless, and can be seen by examining the special shell variable `$?` with:

```
$ echo $?
0
```

The 0 indicates success, i.e., the logic condition succeeded. This approach is especially useful when used inside a shell script, described next.

3 Using uQueryDB within a Shell Script

The motivation for uQueryDB is to support the automation of simulation experiments by launching a mission from within a script and exiting the script (and the mission) when a specified condition is met within the simulation environment. By example we return to the alpha mission. The example script below launches the alpha mission and halts the alpha mission after 20 seconds.

```
#!/bin/bash -e

# Part 1: Launch the mission
pAntler alpha.moos >& /dev/null &

# Part 2: After a short wait deploy the vehicle
sleep 5
uPokeDB alpha.moos MOOS_MANUAL_OVERRIDE=false DEPLOY=true
sleep 5

# Part 3: Continually check for the termination criteria
while true; do
    uQueryDB alpha.moos --condition="DB_UPTIME>30"
    if [ "$?" = 0 ]; then
        echo -n "Quitting..."
        killall pAntler
        sleep 2
        echo "Done."
        exit 0
    fi
    sleep 5
    echo "continuing..."
done
```

Of course the above example simply uses *time* as a termination criteria, which could be accomplished by just changing the `sleep` command and terminating. But `uQueryDB` allows the termination to be based on mission related criteria. For example, in the alpha mission, termination can be set to happen after the vehicle traverses its set of waypoints twice, by setting the logic condition with `--condition="CYCLE_INDEX>2"`.

4 Command-line Arguments of uQueryDB

Configuration parameters can be passed to `uQueryDB` from the command line or from the mission (`.moos`) configuration file. Launching it without logic conditions will simply result in `uQueryDB` returning immediately with a value of 0 (success).

The specification of a MOOS file on the command line is optional. The primary two pieces of information `uQueryDB` needs from this file are (a) the `server_host` IP address, and (b) the `server_port` number of the running MOOSDB to poke. These values can instead be provided on the command line:

```
 $ uQueryDB --condition="NUM>30" --host=18.38.2.158  --port=9000
```

If the `host` or the `port` are not provided on the command line, and a MOOS file is also not provided, the user will be prompted for the two values. Since the most common scenario by convention has the MOOSDB running on the local machine (”localhost”) with port 9000, these are the default values and the user can simply hit the return key.

```
$ uQueryDB --condition="DEPLOY=true"  // User launches with no server host/port info
$ Enter Server: [localhost]           // User accepts default by hitting Return key
$    The server is set to "localhost" // Server host confirmed to be set to "localhost"
$ Enter Port: [9000] 9123             // User overrides the default 9000 port with 9123
$    The port is set to "9123"        // Server port confirmed to be set to "9123"
```

4.1 Logic Pass Conditions on the Command Line

On the command line `uQueryDB` takes any number of logic pass conditions. If *all* pass conditions are satisfied, then `uQueryDB` will return with 0, indicating success. A pass condition may be specified with either the `--condition` or `--pass_condition` parameter. They are interchangeable. A conjuction of pass conditions can be applied simply by specifying mulitiple conditions:

```
 $ uQueryDB alpha.moos --condition="DEPLOY=true" --condition="STATION=false"
```

A disjunction of conditions can be specified as described in http://oceanai.mit.edu/ivpman/logic, within a single condition. For example:

```
 $ uQueryDB alpha.moos --condition="((DB_TIME>36000) or (MISSION=complete))"
```

Since white-space characters on a command line delineate arguments, the use of double-quotes must be used if using white space as in the examples above.

4.2 Logic Fail Conditions on the Command Line

Similar to the pass conditions described above, a *fail* condition may also be specified. Depending on what you're trying to do, it may be just easier to express things with a fail condition(s) instead of pass conditions. If *any* fail condition is satisfied, then `uQueryDB` will return with a value of 1, indicating failure. For example:

```
 $ uQueryDB alpha.moos --fail_condition="DEPLOY=false"   \
                       --fail_condition="STATION=true"
```

Pass and fail conditions may both be used simultaneously. If *all* pass conditions are satisfied, and *no* fail conditions are satisfied, then `uQueryDB` will return with a value of 0, success. Otherwise it will return with a value of 1, failure.

4.3 Waiting for Success

A note of caution: by default, `uQueryDB` will immediately connect to the MOOSDB, evaluate conditions and return with either a 0 or 1. In some cases the user may want to allow `uQueryDB` a period of time before exiting with failure. The `--wait=N` command line parameter may be used, with the default being `N=0`. If `uQueryDB` detects success, it will return immediately. Otherwise, if the pass conditions are not initially all satisfied, or if one or more fail conditions are satisfied initially, `uQueryDB` will continue to evaluate for N seconds before returning with failing (1) value.

4.4 Waiting for a Logic Variable to be Published

Normally a logic condition of the form `(FOO<20)` would evaluate to false if the variable `FOO` has never been written to. This may seem counter-intuitive in conditions like `(DEPLOY!=true)` since clearly `DEPLOY` does not have the value of true if `DEPLOY` has never been written to. But nevertheless the latter condition would also evaluate to false.

5 Check Variables

`uQueryDB` can be optionally configured to report the values of one or more *check variables* at the time that `uQueryDB` exits, regardless of the return value. Check variables are named on either the command line or in the mission configuration. A variety of formats are supported. The motivation for check variables is to enable `uQueryDB` to be further useful in scripts that may be involved in automated mission testing.

5.1 Specifying Check Variables on the Command Line

One or more check variables may be specified on the command line. If variables are also specified in the mission file, the union of all variables will be used.

```
$ uQueryDB --check_var=RESULT --check_var=SCORE
```

5.2 Specifying Check Variables in the Mission File

One or more check variables may be specified in the mission file. If variables are also specified on the command line, the union of all variables will be used.

```
check_var = RESULT
check_var = SCORE
```

5.3 Check Variable Output and Format

The output of check variables is written to a file `.checkvars`. This file will be overwritten upon each run of `uQueryDB`. By default the output will be `equals-separated-value`. For example:

```
RESULT = pass
SCORE = 98
```

Another format option is `comma-separated-value`. For example:

```
RESULT,pass
SCORE,98
```

Another format option is `whitespace-separated-value`. For example:

```
RESULT pass
SCORE 98
```

Another format option is `value-only`. For example:

```
pass
98
```

The format option can be specified on the command line with `--esv` for `equals-separated-value`, or `--csv` for `comma-separated-value`, or `--wsv` for `whitespace-separated-value`, `--vo` for `value-only`.

The format option can also specified in mission file with:

```
check_var_format = esv          // Default setting
check_var_format = wsv
check_var_format = csv
check_var_format = vo
```

If conflicting formats are specified between the command line and the mission file, the mission file setting will take precedence.

6 Configuration Parameters for uQueryDB

In addition to accepting query parameters from the command line, uQueryDB may accept configuration parameters from a configuration block with a `.moos` file, similar to most MOOS applications. This feature was added *after* Release 19.8.x.

The following parameters are defined for uQueryDB. A more detailed description is provided in other parts of this section. Parameters having default values indicate so in parentheses below.

Listing 6.1: Configuration Parameters for uQueryDB.

`check_var`:	When the query results in a return value of zero, an optional variable report can be written to a file. Section 5.
`check_var_format`:	Sets the output format for check variables written to the `.checkvars` file. Options are `esv`, or `csv`, or `wsv`, or `vo`. The default is `esv`. 5.3.
`condition`:	Same as `pass_condition`. When *all* pass conditions are met, the query will result in a return value of zero. Section 4.1.
`fail_condition`:	When *any* fail condition is unsatisfied, the query will result ina return value of zero. Section 4.2.
`pass_condition`:	When *all* pass conditions are met, the query will result in a return value of zero. Section 4.1.
`wait`:	If uQueryDB initially fails, rather than returning with a value of 1 immediatly, it will wait N seconds to receive new mail updates that may perhaps result in successfully satisfying conditions. Default is 0 seconds. Section 4.3.

7 Publications and Subscriptions for uQueryDB

Variables published by the uQueryDB application

The uQueryDB application publishes no variables. In fact there will likely be no trace at all in a mission alog file, with the possible exception being the MOOS variable DB_CLIENTS. If uQueryDB does not immediately find the values for all variables involved in the logic expression, it will stay connected for as long as the --wait parameter specifies. The default is 0 seconds. In this case uQueryDB will be logged in the list of clients in DB_CLIENTS during this period.

Variables subscribed for by the uPokeDB application

All MOOS variables involved in the logic conditions provided to uQueryDB will be subscribed for automatically.