uTimerScript: Scripting Events to the MOOSDB

June 2018

Michael Benjamin, mikerb@mit.edu
Department of Mechanical Engineering
MIT, Cambridge MA 02139
`project-pavlab/appdocs/app_utscript`

1 Overview

The `uTimerScript` application allows the user to script a set of pre-configured posts to a `MOOSDB`. In its most basic form, it may be used to initialize a set of variables to the `MOOSDB`, and immediately

terminate itself if a quit event is included. The following configuration block, if placed in the alpha example mission, would mimic the posts to the MOOSDB behind the DEPLOY button, simply disabling manual control, deploying the vehicle and quitting the script: Listing 1.1.

Listing 1.1: A Simple Timer Script.

```
1  ProcessConfig = uTimerScript
2  {
3    event = var=MOOS_MANUAL_OVERIDE, val=false
4    event = var=DEPLOY, val=true
5    event = quit
6  }
```

Additionally, uTimerScript may be used with the following advanced functions:

- Each entry in the script may be scheduled to occur after a specified amount of elapsed time.
- Event timestamps may be given as an exact point in time relative to the start of the script, or a range in times with the exact time determined randomly at run-time.
- The execution of the script may be paused, or fast-forwarded a given amount of time, or forwarded to the next event on the script by writing to a MOOS variable.
- The script may be conditionally paused based on user defined logic conditions over one or more MOOS variables.
- The variable value of an event may also contain information generated randomly.
- The script may be reset or repeated any given number of times.
- The script may use its own time warp, which can be made to vary randomly between script executions.

In short, uTimerScript may be used to effectively simulate the output of other MOOS applications when those applications are not available. A few examples are provided, including a simulated GPS unit and a crude simulation of wind gusts.

2 Using uTimerScript

Configuring a script minimally involves the specification of one or more events, with an event comprising of a MOOS variable and value to be posted and an optional time at which it is to be posted. Scripts may also be reset on a set policy, or from a trigger by an external process.

2.1 Configuring the Event List

The event list or script is configured by declaring a set of event entries with the following format:

```
event = var=<MOOSVar>, val=<value>, [time=<time-of-event>]
```

The keywords event, var, val, and time are not case sensitive, but the values <moos-variable> and <var-value> are case sensitive. The <var-value> type is posted either as a string or double based on the following heuristic: if the <var-value> has a numerical value it is posted as a double, and

otherwise posted as a string. If one wants to post a string with a numerical value, putting quotes around the number suffices to have it posted as a string. Thus `val=99` posts a double, but `var="99"` posts a string. If a string is to be posted that contains a comma such as `"apples, pears"`, one must put the quotes around the string to ensure the comma is interpreted as part of `<var-value>`. The value field may also contain one or more macros expanded at the time of posting, as described in Section 4.

2.2 Setting the Event Time or Range of Event Times

The value of `<time-of-event>` is given in seconds and must be a numerical value greater or equal to zero. The time represents the amount of elapsed time since the uTimerScript was first launched and un-paused. The list of events provided in the configuration block need not be in order - they will be ordered by the uTimerScript utility. The `<time-of-event>` may also be specified by a interval of time, e.g., `time=0:100`, such that the event may occur at some point in the range with uniform probability. The only restrictions are that the lower end of the interval is greater or equal to zero, and less than or equal to the higher end of the interval. By default the timestamps are calculated once from their specified interval, at the the outset of uTimerScript. The script may alternatively be configured to recalculate the timestamps from their interval each time the script is reset, by setting the `shuffle` parameter to true. This parameter, and resetting in general, are described in the next Section 2.3.

2.3 Resetting the Script

The timer script may be reset to its initial state, resetting the stored elapsed-time to zero and marking all events in the script as pending. This may occur either by cueing from an event outside uTimerScript, or automatically from within uTimerScript. Outside-cued resets can be triggered by posting UTS_RESET with the value `="reset"`, or `"true"`. The `reset_var` parameter names a MOOS variable that may be used as an alternative to UTS_RESET. It has the format:

```
reset_var = <moos-variable> // Default is UTS_RESET
```

The script may be also be configured to auto-reset after a certain amount of time, or immediately after all events are posted, using the `reset_time` parameter. It has the format:

```
reset_time = <time-or-condition> // Default is "none"
```

The `<time-or-condition>` may be set to `"all-posted"` which will reset after the last event is posted. If set to a numerical value greater than zero, it will reset after that amount of elapsed time, regardless of whether or not there are pending un-posted events. If set to `"none"`, the default, then no automatic resetting is performed. Regardless of the `reset_time` setting, prompted resets via the UTS_RESET variable may take place when cued.

The script may be configured to accept a hard limit on the number of times it may be reset. This is configured using the `reset_max` parameter and has the following format:

```
reset_max = <amount> // Default is "nolimit"
```

The `<amount>` specified may be any number greater or equal to zero, where the latter, in effect, indicates that no resets are permitted. If unlimited resets are desired (the default), the case

insensitive argument "unlimited" or "any" may be used.

The script may be configured to recalculate all event timestamps specified with a range of values whenever the script is reset. This is done with the following parameter:

```
shuffle = false // Default is "true"
```

The script may be configured to reset or restart each time it transitions from a situation where its conditions are not met to a situation where its conditions are met, or in other words, when the script is "awoken". The use of logic conditions is described in more detail in Section 3.1. This is done with the following parameter:

```
upon_awake = restart // Default is "n/a", no action
```

Note that this does not apply when the script transitions from being paused to un-paused as described in Section 3.1. See the example in Section 9.2 for a case where the upon_awake feature is handy.

3 Script Flow Control

The script flow may be affected in a number of ways in addition to the simple passage of time. It may be (a) paused by explicitly pausing it, (b) implicitly paused by conditioning the flow on one or more logic conditions, (c) fast-forwarded directly to the next scheduled event, or fast-forwarded some number of seconds. Each method is described in this section.

3.1 Pausing the Timer Script

The script can be paused at any time and set to be paused initially at start time. The paused parameter affects whether the timer script is actively unfolding at the outset of launching uTimerScript. It has the following format:

```
paused = <Boolean>
```

The keyword paused and the string representing the Boolean are not case sensitive. The Boolean simply must be either "true" or "false". By setting paused to true, the elapsed time calculated by uTimerScript is paused and no variable-value pairs will be posted. When un-paused the elapsed time begins to accumulate and the script begins or resumes unfolding. The default value of paused is false.

The script may also be paused through the MOOS variable UTS_PAUSE which may be posted by some other MOOS application. The values recognized are "true", "false", or "toggle", all case insensitive. The name of this variable may be substituted for a different one with the pause_var parameter in the uTimerScript configuration block. It has the format:

```
pause_var = <MOOSVar> // Default is UTS_PAUSE
```

If multiple scripts are being used (with multiple instances of uTimerScript connected to the MOOSDB), setting the pause_var to a unique variable may be needed to avoid unintentionally pausing or un-pausing multiple scripts with single write to UTS_PAUSE.

3.2 Conditional Pausing of the Timer Script and Atomic Scripts

The script may also be configured to condition the "paused-state" to depend on one or more logic conditions. If conditions are specified in the configuration block, the script must be both un-paused as described above in Section 3.1, and all specified logic conditions must be met in order for the script to begin or resume proceeding. The logic conditions are configured as follows:

```
condition = <logic-expression>
```

The `<logic-expression>` syntax is described in Logic Appendix, and may involve the simple comparison of MOOS variables to specified literal values, or the comparison of MOOS variables to one another. See the script configuration in Section 9.2 for one example usage of logic expressions.

An *atomic* script is one that does not check conditions once it has posted its first event, and prior to posting its last event. Once a script has started, it is treated as unpausable with respect to the the logic conditions. This is configured with:

```
script_atomic = <Boolean>
```

It can however be paused and unpaused via the pause variable, e.g., `UTS_PAUSE`, as described in Section 3.1. If the logic conditions suddenly fail in an atomic script midway, the check is simply postponed until after the script completes and is perhaps reset. If the conditions in the meanwhile revert to being satisfied, then no interruption should be observable.

3.3 Fast-Forwarding the Timer Script

The timer script, when un-paused, moves forward in time with events executed as their event times arrive. However, the script may be moved forwarded by writing to the MOOS variable `UTS_FORWARD`. If the value received is zero (or negative), the script will be forwarded directly to the point in time at which the next scheduled event occurs. If the value received is positive, the elapsed time is forwarded by the given amount. Alternatives to the MOOS variable `UTS_FORWARD` may be configured with the parameter:

```
forward_var = <MOOSVar> // Default is UTS_FORWARD
```

If multiple scripts are being used (with multiple instances of `uTimerScript` connected to the `MOOSDB`), setting the `forward_var` to a unique variable may be needed to avoid unintentionally fast forwarding multiple scripts with single write to `UTS_FORWARD`.

3.4 Quitting the Timer Script

The timer script may be configured with a special event, the *quit* event, resulting in disconnection with the `MOOSDB` and a process exit. This is done with the configuration:

```
event = quit [time=<time-of-event>]
```

Before quitting, a final posting to the `MOOSDB` is made with the variable `EXITED_NORMALLY`. The value is `"uTimerScript"`, or its alias if an alias was used. This indicates to any other watchdog process, such as `uProcessWatch`, that the exiting of this script is not a reason for concern. When `uTimerScript` receives its own posting in the next incoming mail, it assumes all pending posts have been made and will then quit.

4 Macro Usage in Event Postings

Macros may be used to add a dynamic component to the value field of an event posting. This substantially expands the expressive power and possible uses of the uTimerScript utility. Recall that the components of an event are defined by:

```
event = var=<MOOSVar>, val=<var-value>, time=<time-of-event>
```

The <var-value> component may contain a macro of the form $[MACRO], where the macro is either one of a few built-in macros available, or a user-defined macro with the ability to represent random variables. Macros may also be combined in simple arithmetic expressions to provide further expressive power. In each case, the macro is expanded at the time of the event posting, typically with different values on each successive posting.

4.1 Built-In Macros Available

There are five built-in macros available: $[DBTIME], $[UTCTIME], $[COUNT], $[TCOUNT], and $[IDX]. The first macro expands to the estimated time since the MOOSDB started, similar to the value in the MOOS variable DB_UPTIME published by the MOOSDB. An example usage:

```
event = var=DEPLOY_RECEIVED, val=$[DBTIME], time=10:20
```

The $[UTCTIME] macro expands to the UTC time at the time of the posting. The $[COUNT] macro expands to the integer total of all posts thus far in the current execution of the script, and is reset to zero when the script resets. The $[TCOUNT] macro expands to the integer total of all posts thus far since the application began, i.e., it is a running total that is not reset when the script is reset.

The $[DBTIME], $[UTCTIME], $[COUNT], and $[TCOUNT] macros all expand to numerical values, which if embedded in a string, will simply become part of the string. If the value of the MOOS variable posting is solely this macro, the variable type of the posting is instead a double, not a string. For example val=$[DBTIME] will post a type double, whereas val="time:$[DBTIME]" will post a type string.

The $[IDX] macro is similar to the $[COUNT] macro in that it expands to the integer value representing an event's count or index into the sequence of events. However, it will always post as a string and will be padded with zeros to the left, e.g., "000", "001", ... and so on.

4.2 User Configured Macros with Random Variables

Further macros are available for use in the <var-value> component of an event, defined and configured by the user, and based on the idea of a random variable. In short, the macro may expand to a numerical value chosen within a user specified range, and recalculated according to a user-specified policy. The general format is:

```
rand_var = varname=<variable>, min=<value>, max=<value>, key=<key_name>
```

The <variable> component defines the macro name. The <low_value> and <high_value> components define the range from which the random value will be chosen uniformly. The <key_name> determines when the random value is reset. It must be set to one of the following three values: "at_start",

"at_reset", and "at_post". Random variables with the key name "at_start" are assigned a random value only at the start of the uTimerScript application. Those with the "at_reset" key name also have their values re-assigned whenever the script is reset. Those with the "at_post" key name also have their values re-assigned after any event is posted.

4.3 Support for Simple Arithmetic Expressions with Macros

Macros that expand to numerical values may be combined in simple arithmetic expressions with other macros or scalar values. The general form is:

```
{<value> <operator> <value>}
```

The <value> components may be either a scalar or a macro, and the <operator> component may be one of '+', '-', '*', '/'. Nesting is also supported. Below are some examples:

```
{$[FOOBAR] * 0.5}
{-2-$[FOOBAR]}
{$[APPLES] + $[PEARS]}
{35 / {$[FOOBAR]-2}}
{$[DBTIME] - {35 / {$[UTCTIME]+2}}}
```

If a macro should happen to expand to a string rather than a double (numerical) value, the string evaluates to zero for the sake of the remaining evaluations.

5 Time Warps, Random Time Warps, and Restart Delays

A time warp and initial start delay may be optionally configured into the script to change the event schedule without having to edit all the time entries for each event. They may also be configured to take on a new random value at the outset of each script execution to allow for simulation of events in nature or devices having a random component.

5.1 Random Time Warping

The time warp is a numerical value in the range $(0, \infty]$, with a default value of 1.0. Lower values indicate that time is moving more slowly. As the script unfolds, a counter indicating "elapsed_time" increases in value as long as the script is not paused. The "elapsed_time" is multiplied by the time warp value. The time warp may be specified as a single value or a range of values as below:

```
time_warp = <value>
time_warp = <low-value>:<high-value>
```

When a range of values is specified, the time warp value is calculated at the outset, and re-calculated whenever the script is reset. See the example in Section 9.3 for a use of random time warping to simulate random wind gusts.

5.2 Random Initial Start and Reset Delays

A start delay may be provided with the delay_start parameter, given in seconds in the range $[0, \infty]$, with a default value of 0. The effect of having a non-zero delay of n seconds is to have elapsed_time=n at the outset of the script, on the first time through the script only. Thus a delay of n seconds combined with a time warp of 0.5 would result in observed delay of $2 * n$ seconds. The start delay may be specified as a single value or a range of values as below:

```
delay_start = <value>
delay_start = <low-value>:<high-value>
```

To specify a delay applied at the beginning if the script *after a reset*, use the delay_reset parameter instead.

```
delay_reset = <value>
delay_reset = <low-value>:<high-value>
```

When a range of values is specified, the start ore reset delay value is calculated at the outset, and re-calculated whenever the script is reset. See the example in Section 9.2 for a use of random start delays to the simulate the delay in acquiring satellite fixes in a GPS unit on an UUV coming to the surface.

5.3 Status Messages Posted to the MOOSDB by uTimerScript

The uTimerScript periodically publishes a string to the MOOS variable UTS_STATUS indicating the status of the script. This variable will be published on each iteration if one of the following conditions is met: (a) two seconds has passed since the previous status message posted, or (b) an event has been been posted, or (c) the paused state has changed, or (d) the script has been reset, or (e) the state of script logic conditions has changed. A posting may look something like:

```
UTS_STATUS = "name=RND_TEST, elapsed_time=2.00, posted=1, pending=5, paused=false,
              conditions_ok=true, time_warp=3, start_delay=0, shuffle=false,
              upon_awake=restart, resets=2/5"
```

In this case, the script has posted one of six events (posted=1, pending=5). It is actively unfolding, since paused=false (Section 3.1) and conditions_ok=true (Section 3.2). It has been reset twice out of a maximum of five allowed resets (resets=2/5, Section 2.3). Time warping is being deployed time_warp=3 (Section 5), there is no start delay in use start_delay=0 (Section 5.2). The shuffle feature is turned off shuffle=false (Section 2.3). The script is not configured to reset upon re-entering the un-paused state, awake_reset=false (Section 2.3).

When multiple scripts are running in the same MOOS community, one may want to take measures to discern between the status messages generated across scripts. One way to do this is to use a unique MOOS variable other than UTS_STATUS for each script. The variable used for publishing the status may be configured using the status_var parameter. It has the following format:

```
status_var = <MOOSVar> // Default is UTS_STATUS
```

Alternatively, a unique name may be given to each to each script. All status messages from all scripts would still be contained in postings to UTS_STATUS, but the different script output could be discerned by the name field of the status string. The script name is set with the following format.

```
script_name = <string> // Default is "unnamed"
```

6 Terminal and AppCast Output

The script configuration and progress of script execution may also be monitored from an open console window where uTimerScript is launched, or through an appcast viewer. Example output is shown below in Listing 2. On line 2, the name of the local community or vehicle name is listed on the left. On the right, "0/0(450) indicates there are no configuration or run warnings, and the current iteration of uFldTimerScript is 450.

Lines 4-16: Script Configuration

Lines 4-11 show the script configuration. Line 5 shows the number of elements in the script and in parentheses the last element to have been posted. Line 6 shows the number of times the script has restarted. Line 7 shows the present time warp and the range of time warps possible on each script restart in brackets (Section 5.1. Lines 8-9 show the delay applied at the start and after a script reset (Section 5.2). Line 10 indicates the script is presently not paused (Section 3.1). Line 11 indicates the script presently meets any prevailing logic conditions (Section 3.2). Lines 13-16 show that there are two random variables defined for this script that may be used in event definitions. They are both uniform random variables. The first varies over possible directions, and the second over possible speed magnitudes. Section 4.2.

Listing 6.2: Example uTimerScript console and appcast output.

```
1  ============================================================
2  uTimerScript charlie                                 0/0(450)
3  ============================================================
4  Current Script Information:
5      Elements: 10(8)
6       Reinits: 2
7     Time Warp: 1.09 [0.2,2]
8   Delay Start: 0
9   Delay Reset: 23.66 [10,60]
10        Paused: false
11  ConditionsOK: true
12
13  RandomVar  Type     Min  Max  Parameters
14  ---------  -------  ---  ---  ----------
15  ANG        uniform  0    359
16  MAG        uniform  1.5  3.5
17
18  P/Tot  P/Loc  T/Total   T/Local  Variable/Var
19  -----  -----  --------  -------  ------------
20  19     9      196.77    72.15    DRIFT_VECTOR_ADD = 193,-0.6
21  20     0      219.42    24.08    DRIFT_VECTOR_ADD = 12,0.4
22  21     1      220.93    25.71    DRIFT_VECTOR_ADD = 12,0.4
```

```
23  22     2      222.95    27.91    DRIFT_VECTOR_ADD = 12,0.4
24  23     3      224.96    30.09    DRIFT_VECTOR_ADD = 12,0.4
25  24     4      226.46    31.73    DRIFT_VECTOR_ADD = 12,0.4
26  25     5      228.47    33.91    DRIFT_VECTOR_ADD = 12,-0.4
27  26     6      230.49    36.10    DRIFT_VECTOR_ADD = 12,-0.4
28
29  ================================================================
30  Most Recent Events (3):
31  ================================================================
32  [192.78]: Script Re-Start. Warp=0.48922, DelayStart=0.0, DelayReset=54.1
33  [44.80]: Script Re-Start. Warp=1.61692, DelayStart=0.0, DelayReset=18.9
34  [0.51]: Script Start. Warp=1, DelayStart=0.0, DelayReset=0.0
```

Lines 18-27: Recent Script Postings

Lines 18-27 show recent postings to the MOOSDB by the script. The first column shows the total postings so far for the script. The second column shows the index within the script. In the above example, there are ten elements in the script. The most recent posting on line 27, shows the script has been reset twice and the most recent posting is of the seventh element of the script (index 6). The third column shows the total time since script started, and the fourth column shows the time since the script was re-started. Note the time delay between lines 20 and 21, due to the delay_reset shown on line 9. The last column shows the actual variable value pair posted.

Lines 29-34: Recent Events

Lines 29-34 show recent events (other than event postings). In this case it shows the script has been started, and re-started twice. Notice the delay reset on line 32 is different than that on line 9. The delay reset time of 23.66 seconds shown on line 9 is the delay reset to be applied on the *next* reset.

7 Configuration Parameters for uTimerScript

The following parameters are defined for uTimerScript. A more detailed description is provided in other parts of this section. Parameters having default values are indicated.

Listing 7.3: Configuration Parameters for uTimerScript.

block_on:	A comma-separated list of MOOS apps on which the script will block until seen in the list of DB_CLIENTS.
condition:	A logic condition that must be met for the script to be un-paused. Section 3.2.
delay_reset:	Number of seconds added to each event time, on each script reset. Legal values: any non-negative numerical value, or range of values separated by a colon. The default is zero. Section 5.2.
delay_start:	Number of seconds, or range of seconds, added to each event time, on first pass only. Legal values: any non-negative numerical value, or range of values separated by a colon. The default is zero. Section 5.2.
event:	A description of a single event in the timer script. Section 2.1.

forward_var: A MOOS variable for taking cues to forward time. The default is UTS_FORWARD). Section 3.3.

paused: A Boolean indicating whether the script is paused upon launch. Legal values: true, false. The default is false. Section 3.1.

pause_var: A MOOS variable for receiving pause state cues (UTS_PAUSE). Section 3.1.

rand_var: A declaration of a random variable macro to be expanded in event values. Section 4.2.

reset_max: The maximum amount of resets allowed. Legal values: any non-negative integer, or the string "nolimit". The default is "nolimit". NOTE: If resetting is desired, the reset_time parameter must also be changed from its default value of "none". Section 2.3.

reset_time: The time or condition when the script is reset Legal values: Any non-negative number, the string "none", or "all-posted", or equivalently, "end". The default is "none". Section 2.3.

reset_var: A MOOS variable for receiving reset cues. The default is UTS_RESET.

script_atomic: When true, a started script will complete if conditions suddenly fail. Legal values: true, false. The default is false.

script_name: Unique (hopefully) name given to this script. The default is "unnamed".

shuffle: If true, timestamps are recalculated on each reset of the script. Legal values: true, false. The default is true. Section 2.3.

status_var: A MOOS variable for posting status summary. The default is UTS_STATUS. Section 5.3

time_warp: Rate at which time is accelerated in executing the script. Legal values: any non-negative number. The default is zero. Section 5.

time_zero: The base time upon which script event times are based. The default is "script_start", the time at which uTimerScript is launched. The one alternative is "db_start", the time at which the MOOSDB was launched. The default is "script_start".

upon_awake: Reset or re-start the script upon conditions being met after failure ("n/a"). Section 2.3.

verbose: If true, progress output is generated to the console (true).

8 Publications and Subscriptions for uTimerScript

The interface for uTimerScript, in terms of publications and subscriptions, is described below. This same information may also be obtained from the terminal with:

```
$ uTimerScript --interface or -i
```

8.1 Variables Published by uTimerScript

The primary output of uTimerScript to the MOOSDB is the set of configured events, but one other variable is published on each iteration, and another upon purposeful exit with the event=quit event

configuration.

- APPCAST: Contains an appcast report identical to the terminal output. Appcasts are posted only after an appcast request is received from an appcast viewing utility. Section 6.
- EXITED_NORMALLY: A posting made when the script contains and executes a event=quit event, to let other applications know that the disconnection of uTimerScript is not a concern for alarm.
- UTS_STATUS: A status string of script progress. Section 5.3.

8.2 Variables Subscribed for by uTimerScript

The uTimerScript application will subscribe for the following four MOOS variables to provide optional control over the flow of the script by the user or other MOOS processes:

- APPCAST_REQ: A request to generate and post a new apppcast report, with reporting criteria, and expiration.
- EXITED_NORMALLY: When uTimerScript receives its own posting, it is assumed that all outgoing posts needed to be made before quitting have been received by the MOOSDB. Upon this receipt uTimerScript will quit. See Section 3.4.
- UTS_NEXT: When received with the value "next", the script will fast-forward in time to the next event. See Section 3.3.
- UTS_RESET: When received with the value of either "true" or "reset", the timer script will be reset. See Section 2.3.
- UTS_FORWARD: When received with a numerical value greater than zero, the script will fast-forward by the indicated time. See Section 3.3.
- UTS_PAUSE: When received with the value of "true", "false", "toggle", the script will change its pause state correspondingly. See Section 3.1.

In addition to the above MOOS variables, uTimerScript will subscribe for any variables involved in logic conditions, described in Section 3.2.

8.3 An Example MOOS Configuration Block

To see an example MOOS configuration block, enter the following from the command-line:

```
$ uTimerScript --example
```

This will show the output shown in Listing 4 below.

Listing 8.4: Example configuration of the uTimerScript application.

```
1  ============================================================
2  uTimerScript Example MOOS Configuration
3  ============================================================
4  Blue lines:     Default configuration
5
6  ProcessConfig = uTimerScript
```

```
 7  {
 8    AppTick   = 4
 9    CommsTick = 4
10
11    // Logic condition that must be met for script to be unpaused
12    condition      = WIND_GUSTS = true
13    // Seconds added to each event time, on each script pass
14    delay_reset    = 0
15    // Seconds added to each event time, on first pass only
16    delay_start    = 0
17    // Event(s) are the key components of the script
18    event          = var=SBR_RANGE_REQUEST, val="name=archie", time=25:35
19    // A MOOS variable for taking cues to forward time
20    forward_var    = UTS_FORWARD   // or other MOOS variable
21    // If true script is paused upon launch
22    paused         = false    // or {true}
23    // A MOOS variable for receiving pause state cues
24    pause_var      = UTS_PAUSE     // or other MOOS variable
25    // Declaration of random var macro expanded in event values
26    randvar        = varname=ANG, min=0, max=359, key=at_reset
27    // Maximum number of resets allowed
28    reset_max      = nolimit  // or in range [0,inf)
29    // A point when the script is reset
30    reset_time     = none     // or {all-posted} or range (0,inf)
31    // A MOOS variable for receiving reset cues
32    reset_var      = UTS_RESET     // or other MOOS variable
33    // If true script will complete if conditions suddenly fail
34    script_atomic  = false    // or {true}
35    // A hopefully unique name given to the script
36    script_name    = unnamed
37    // If true timestamps are recalculated on each script reset
38    shuffle        = true
39    // If true progress is generated to the console
40    verbose        = true     // or {false}
41    // Reset or restart script upon conditions being met after failure
42    upon_awake     = n/a      // or {reset,resstart}
43    // A MOOS variable for posting the status summary
44    status_var     = UTS_STATUS    // or other MOOS variable
45    // Rate at which time is accelerated in execuing the script
46    time_warp      = 1
47  }
```

9 Examples

The examples in this section demonstrate the constructs thus far described for the uTimerScript application. In each case, the use of the script obviated the need for developing and maintaining a separate dedicated MOOS application.

9.1 A Script for Generating 100 Random Numbers

The below script will generate 100 postings with random numbers preceded by an initial "start" posting, and followed by an "end" posting.

Listing 9.5: A uTimerScript configuration for generating 100 random numbers.

```
//------------------------------------------
ProcessConfig = uTimerScript
{
  rand_var    = varname=RND_VAL, min=0, max=50, key=at_post

  event = var=REPORT, val="start", time=0
  event = var=REPORT, val="accidents=$[RND_VAL], unique_id=$[TCOUNT]", time=0, amt=100
  event = var=REPORT, val="end", time=0
}
```

The above may result in postings like those below (take from an alog file):

```
8.817     REPORT         uTimerScript    start
8.817     REPORT         uTimerScript    accidents=35.234,unique_id=1
8.817     REPORT         uTimerScript    accidents=32.01,unique_id=2
8.818     REPORT         uTimerScript    accidents=26.982,unique_id=3
...
8.856     REPORT         uTimerScript    accidents=32.3,unique_id=95
8.857     REPORT         uTimerScript    accidents=13.324,unique_id=96
8.857     REPORT         uTimerScript    accidents=27.624,unique_id=97
8.858     REPORT         uTimerScript    accidents=28.901,unique_id=98
8.858     REPORT         uTimerScript    accidents=18.9,unique_id=99
8.859     REPORT         uTimerScript    accidents=35.72,unique_id=100
8.859     REPORT         uTimerScript    end
```

To make the numerical values integers, use the `snap=1` option (available in the next release after 19.8). Using `snap=0.1` will round to the nearest tenth and so on. For example:

```
  rand_var    = varname=RND_VAL, min=0, max=50, key=at_post, snap=1
```

9.2 A Script Used as Proxy for an On-Board GPS Unit

Typical operation of an underwater vehicle includes the periodic surfacing to obtain a GPS fix to correct navigation error accumulated while under water. A GPS unit that has been out of satellite communication for some period normally takes some time to re-acquire enough satellites to resume providing position information. From the perspective of the helm and configuring an autonomy mission, it is typical to remain at the surface only long enough to obtain the GPS fix, and then resume other aspects of the mission at-depth.

Consider a situation as shown in Figure 1, where the autonomy system is running in the payload on a payload computer, receiving not only updated navigation positions (in the form of `NAV_DEPTH`, `NAV_X`, and `NAV_Y`), but also a "heartbeat" signal each time a new GPS position has been received (`GPS_RECEIVED`). This heartbeat signal may be enough to indicate to the helm and mission configuration that the objective of the surface excursion has been achieved.

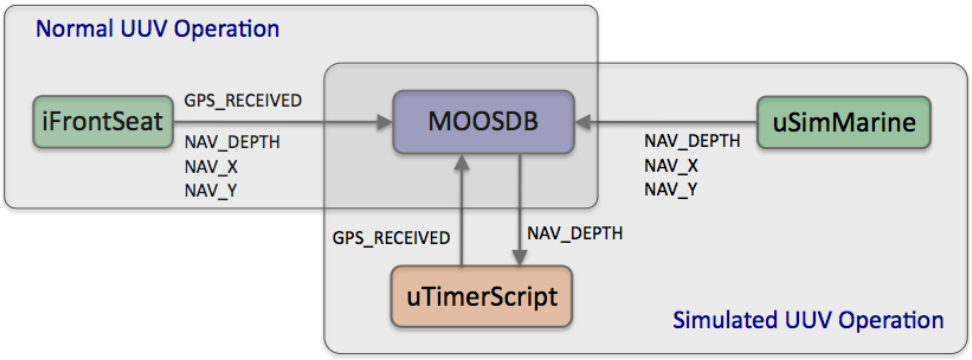

Figure 1: **Simulating a GPS Acknowledgment:** In a physical operation of the vehicle, the navigation solution and a GPS_UPDATE_RECEIVED heartbeat are received from the main vehicle (front-seat) computer via a MOOS module acting as an interface to the front-seat computer. In simulation, the navigation solution is provided by the simulator without any GPS_UPDATE_RECEIVED heartbeat. This element of simulation may be provided with uTimerScript configured to post the heartbeat, conditioned on the NAV_DEPTH information and a user-specified start delay to simulate GPS acquisition delay.

In simulation, however, the simulator only produces a steady stream of navigation updates with no regard to a simulated GPS unit. At this point there are three choices: (a) modify the simulator to fake GPS heartbeats and satellite delay, (b) write a separate simple MOOS application to do the same simulation. The drawback of the former is that one may not want to branch a new version of the simulator, or even introduce this new complexity to the simulator. The drawback of the latter is that, if one wants to propagate this functionality to other users, this requires distribution and version control of a new MOOS application.

A third and perhaps preferable option (c) is to write a short script for uTimerScript simulating the desired GPS characteristics. This achieves the objectives without modifying or introducing new source code. The below script in Listing 6 gets the job done.

Listing 9.6: A uTimerScript configuration for simulating aspects of a GPS unit.

```
1  //------------------------------------------------------------
2  // uTimerScript configuration block
3
4  ProcessConfig = uTimerScript
5  {
6    AppTick   = 4
7    CommsTick = 4
8
9    paused      = false
10   reset_max   = unlimited
11   reset_time  = end
12   condition   = NAV_DEPTH < 0.2
13   upon_awake  = restart
14   delay_start = 20:120
15   script_name = GPS_SCRIPT
16
17   event  = var=GPS_UPDATE_RECEIVED, val="RCVD_$[COUNT]", time=0:1
18 }
```

This script posts a GPS_UPDATE_RECEIVED heartbeat message roughly once every second, based on the event time "time=0:1" on line 17. The value of this message will be unique on each posting due to the $[COUNT] macro in the value component. See Section 4.1 for more on macros. The script is configured to restart each time it awakes (line 13), defined by meeting the condition of (NAV_DEPTH < 0.2) which is a proxy for the vehicle being at the surface. The delay_start simulates the time needed for the GPS unit to reacquire satellite signals and is configured to be somewhere in the range of 20 to 120 seconds (line 14). Once the script gets past the start delay, the script is a single event (line 17) that repeats indefinitely since reset_max is set to unlimited and reset_time is set to end in lines 10 and 11. This script is used in the IvP Helm example simulation mission labeled "s4_delta" illustrating the PeriodicSurface helm behavior.

9.3 A Script as a Proxy for Simulating Random Wind Gusts

Simulating wind gusts, or in general, somewhat random external periodic drift effects on a vehicle, are useful for testing the robustness of certain autonomy algorithms. Often they don't need to be grounded in very realistic models of the environment to be useful, and here we show how a script can be used simulate such drift effects in conjunction with the uSimMarine application.

The uSimMarine application is a simple simulator that produces a stream of navigation information, NAV_X, NAV_Y, NAV_SPEED, NAV_DEPTH, and NAV_HEADING (Figure 2), based on the vehicle's last known position and trajectory, and currently observed values for actuator variables. The simulator also stores local state variables reflecting the current external drift in the x-y plane, by default zero. An external drift may be specified in terms of a drift vector, in absolute terms with the variable USM_DRIFT_VECTOR, or in relative terms with the variables USM_DRIFT_VECTOR_ADD.

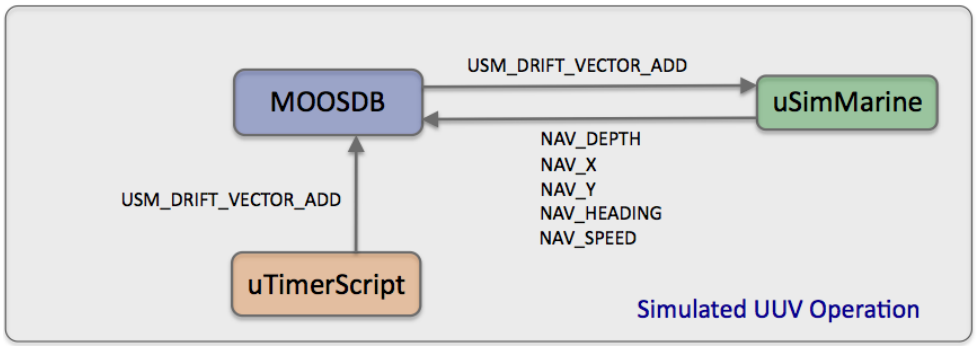

Figure 2: **Simulated Wind Gusts:** The uTimerScript application may be configured to post periodic sequences of external drift values, used by the uSimMarine application to simulate wind gust effects on its simulated vehicle.

The script in Listing 7 makes use of the uSimMarine interface by posting periodic drift vectors. It simulates a wind gust with a sequence of five posts to increase a drift vector (lines 18-22), and complementary sequence of five posts to decrease the drift vector (lines 24-28) for a net drift of zero at the end of each script execution.

Listing 9.7: A uTimerScript configuration for simulating simple wind gusts.

```
1  //------------------------------------------
```

```
 2  // uTimerScript configuration  block
 3
 4  ProcessConfig = uTimerScript
 5  {
 6    AppTick   = 2
 7    CommsTick = 2
 8
 9    paused        = false
10    reset_max     = unlimited
11    reset_time    = end
12    delay_reset   = 10:60
13    time_warp     = 0.25:2.0
14    script_name   = WIND
15    script_atomic = true
16
17    randvar = varname=ANG, min=0,   max=359, key=at_reset
18    randvar = varname=MAG, min=0.5, max=1.5, key=at_reset
19
20  event = var=DRIFT_VECTOR_ADD, val="$(ANG),{$(MAG)*0.2}", time=0
21  event = var=DRIFT_VECTOR_ADD, val="$(ANG),{$(MAG)*0.2}", time=2
22  event = var=DRIFT_VECTOR_ADD, val="$(ANG),{$(MAG)*0.2}", time=4
23  event = var=DRIFT_VECTOR_ADD, val="$(ANG),{$(MAG)*0.2}", time=6
24  event = var=DRIFT_VECTOR_ADD, val="$(ANG),{$(MAG)*0.2}", time=8
25
26  event = var=DRIFT_VECTOR_ADD, val="$(ANG),{$(MAG)*-0.2}", time=10
27  event = var=DRIFT_VECTOR_ADD, val="$(ANG),{$(MAG)*-0.2}", time=12
28  event = var=DRIFT_VECTOR_ADD, val="$(ANG),{$(MAG)*-0.2}", time=14
29  event = var=DRIFT_VECTOR_ADD, val="$(ANG),{$(MAG)*-0.2}", time=16
30  event = var=DRIFT_VECTOR_ADD, val="$(ANG),{$(MAG)*-0.2}", Time=18
31  }
```

The drift *angle* is chosen randomly in the range of $[0, 359]$ by use of the random variable macro `$[ANG]` defined on line 16. The peak *magnitude* of the drift vector is chosen randomly in the range of $[0.5, 1.5]$ with the random variable macro `$[MAG]` defined on line 17. Note that these two macros have their random values reset each time the script begins, by using the `key=at_reset` option, to ensure a stream of wind gusts of varying angles and magnitudes.

The duration of each gust sequence also varies between each script execution. The default duration is about 20 seconds, given the timestamps of 0 to 18 seconds in lines 19-29. The `time_warp` option on line 12 affects the duration with a random value chosen from the interval of $[0.25, 2.0]$. A time warp of 0.25 results in a gust sequence lasting about 80 seconds, and 2.0 results in a gust of about 10 seconds. The time between gust sequences is chosen randomly in the interval $[10, 60]$ by use of the `delay_restart` parameter on line 11. Used in conjunction with the `time_warp` parameter, the interval for possible observed delays between gusts is $[5, 240]$. The `reset_time` parameter set to `end`, on line 10 is used to ensure that the script posts all drift vectors to avoid any accumulated drifts over time. The `reset_max` parameter is set to `"unlimited"` to ensure the script runs indefinitely.