10 The uTimerScript Utility: Scripting Events to the MOOSDB

The uTimerScript application allows the user to script a set of pre-configured pokes to a MOOSDB with each entry in the script happening after a specified amount of elapsed 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 from which uTimerScript accepts such cues. 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 variable value of an event may also contain information generated randomly. The script may also be reset or repeated any given number of times. In short, the uTimerScript application may be used to effectively simulate the output of other MOOS applications when those applications are not available. We give a few examples, including a simulated GPS unit and a crude simulation of wind gusts.

10.1 Overview of the uTimerScript Interface and Configuration Options

The uTimerScript application may be configured with a configuration block within a .moos file, and from the command line. Its interface is defined by its publications and subscriptions for MOOS variables consumed and generated by other MOOS applications. An overview of the set of configuration options and interface is provided in this section.

10.1.1 Configuration Parameters uTimerScript

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

- **CONDITION**: A logic condition that must be met for the script to be un-paused.
- **DELAY_RESET**: Number of seconds added to each event time, *on each script pass* (0).
- **DELAY_START**: Number of seconds added to each event time, *on first pass only* (0).
- **EVENT**: A description of a single event in the timer script.
- **FORWARD_VAR**: A MOOS variable for taking cues to forward time (UTS_FORWARD).
- **PAUSED**: A Boolean indicating whether the script is paused upon launch (false).
- **PAUSE_VAR**: A MOOS variable for receiving pause state cues (UTS_PAUSE).
- **RAND_VAR**: A declaration of a random variable macro to be expanded in event values.
- **RESET_MAX**: The maximum amount of resets allowed (*nolimit*).
- **RESET_TIME**: The time or condition when the script is reset (*none*).
- **RESET_VAR**: A MOOS variable for receiving reset cues (UTS_RESET).
- **SCRIPT_ATOMIC**: When true, a started script will complete if conditions suddenly fail (false).
- **SCRIPT_NAME**: Unique (hopefully) name given to this script (*unnamed*).
- **SHUFFLE**: If true, timestamps are recalculated on each reset of the script (true).
- **STATUS_VAR**: A MOOS variable for posting status summary (UTS_STATUS).
- **TIME_WARP**: Rate at which time is accelerated, $[0, \infty]$, in executing the script (1).
- **UPON_AWAKE**: Reset or re-start the script upon conditions being met after failure (*n/a*).
- **VERBOSE**: If true, progress output is generated to the console (true).
10.1.2 MOOS Variables Posted 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:

**UTS_STATUS**: A status string of script progress.

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. An example string:

```plaintext
UTS_STATUS = "name=RND_TEST, elapsed_time=2.00, posted=1, pending=4, paused=false,
    conditions_ok=true, time_warp=3, start_delay=0, shuffle=false, upon_awake=reset, resets=0/4"
```

10.1.3 MOOS 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:

- **UTS_NEXT**: When received with the value "next", the script will fast-forward in
time to the next event. Described in Section 10.3.3.
- **UTS_RESET**: When received with the value of either "true" or "reset", the timer
script will be reset. Described in Section 10.2.2.
- **UTS_FORWARD**: When received with a numerical value greater than zero, the script will
fast-forward by the indicated time. Described in Section 10.3.3.
- **UTS_PAUSE**: When received with the value of "true", "false", "toggle", the script
will change its pause state correspondingly. Described in Section 10.3.1.

In addition to the above MOOS variables, uTimerScript will subscribe for any variables involved
in logic conditions, described in Section 10.3.2.

10.1.4 Command Line Usage of uTimerScript

The uTimerScript application is typically launched as a part of a batch of processes by pAntler,
but may also be launched from the command line by the user. The basic command line usage for
the uTimerScript application is the following:

```
Listing 35 - Command line usage for the uTimerScript tool.
```

```plaintext
0 Usage: uTimerScript file.moos [OPTIONS]
1
2 Options:
3   --alias=<ProcessName>
4     Launch uTimerScript with the given process name
5       rather than uTimerScript.
6   --example, -e
7     Display example MOOS configuration block
8   --help, -h
```
Display this help message.

--shuffle=Boolean (true/false)
If true, script is recalculated on each reset.
If event times configured with random range, the
ordering may change after a reset.
The default is true.

--verbose=Boolean (true/false)
Display script progress and diagnostics if true.
The default is true.

--version,-v
Display the release version of uTimerScript.

Note that the --alias option is the only way to launch more than one uTimerScript process
connected to the same MOOSDB.

10.1.5 An Example MOOS Configuration Block

As of MOOS-IvP Release 4.2, most if not all MOOS apps are implemented to support the -e
or --example command-line switches. To see an example MOOS configuration block, enter the
following from the command-line:

$ uTimerScript -e

This will show the output shown in Listing 36 below.

Listing 36 - Example configuration of the uTimerScript application.

ProcessConfig = uTimerScript
{
  AppTick = 4
  CommsTick = 4
  
  // Logic condition that must be met for script to be unpaused
  condition = WIND_GUSTS = true
  // Seconds added to each event time, on each script pass
  delay_reset = 0
  // Seconds added to each event time, on first pass only
  delay_start = 0
  
  // Event(s) are the key components of the script
  event = var=SBR_RANGE_REQUEST, val="name=archie", time=25:35
  // A MOOS variable for taking cues to forward time
  forward_var = UTS_FORWARD // or other MOOS variable
  // If true script is paused upon launch
  paused = false // or {true}
  // A MOOS variable for receiving pause state cues
  pause_var = UTS_PAUSE // or other MOOS variable
  
  // Declaration of random var macro expanded in event values
}
### 10.2 Basic Usage of the uTimerScript Utility

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 the 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.

#### 10.2.1 Configuring the Event List

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

```
EVENT = var=<moos-variable>, val=<var-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 10.4.

### 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, using the \texttt{SHUFFLE=true} configuration. This parameter, and resetting in general, are described in the next Section (10.2.2).

\subsection*{10.2.2 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 \texttt{uTimerScript}, or automatically from within \texttt{uTimerScript}. Outside-cued resets can be triggered by posting \texttt{UTS\_RESET="reset"}, or "true". The \texttt{RESET\_VAR} parameter names a MOOS variable that may be used as an alternative to \texttt{UTS\_RESET}. It has the format:

\begin{verbatim}
RESET\_VAR = <moos-variable> // Default is UTS\_RESET
\end{verbatim}

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 \texttt{RESET\_TIME} parameter. It has the format:

\begin{verbatim}
RESET\_TIME = <time-or-condition> // Default is "none"
\end{verbatim}

The \texttt{<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 \texttt{RESET\_TIME} setting, prompted resets via the \texttt{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 \texttt{RESET\_MAX} parameter and has the following format:

\begin{verbatim}
RESET\_MAX = <amount> // Default is "nolimit"
\end{verbatim}

The \texttt{<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:

\begin{verbatim}
SHUFFLE = true // Default is "false"
\end{verbatim}

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 10.3.1. This is done with the following parameter:

\begin{verbatim}
UPON\_AWAKE = restart // Default is "n/a", no action
\end{verbatim}

Note that this does not apply when the script transitions from being paused to un-paused as described in Section 10.3.1. See the example in Section 10.7.1 for a case where the \texttt{UPON\_AWAKE} feature is handy.
10.3 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.

10.3.1 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:

\[
\text{PAUSED} = \text{<Boolean>}
\]

The keyword PAUSED and the string representing the Boolean are not case sensitive. The Boolean simply must be either "true" or "false". By setting \text{PAUSED}=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 is \text{PAUSED}=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 \text{PAUSE_VAR} parameter in the uTimerScript configuration block. It has the format:

\[
\text{PAUSE_VAR} = \text{<moos-variable>} \quad \text{// Default is UTS_PAUSE}
\]

If multiple scripts are being used (with multiple instances of uTimerScript connected to the MOOSDB), setting the \text{PAUSE_VAR} to a unique variable may be needed to avoid unintentionally pausing or unpausing multiple scripts with single write to UTS_PAUSE.

10.3.2 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 10.3.1, 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:

\[
\text{CONDITION} = \text{<logic-expression>}
\]

The \text{<logic-expression>} syntax is described in Appendix A, 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 10.7.1 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. It can however be paused and unpaused via the pause variable, e.g., UTS_PAUSE, as described in Section 10.3.1. 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.
10.3.3 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 forward 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 = <moos-variable>  // 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`.

10.4 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=<moos-variable>, 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.

10.4.1 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.

### 10.4.2 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:

\[
\text{RAND\_VAR = varname=\text{variable}, min=\text{low\_value}, max=\text{high\_value}, key=\text{key\_name}}
\]

The \text{variable} component defines the macro name. The \text{low\_value} and \text{high\_value} components define the range from which the random value will be chosen uniformly. The \text{key\_name} determines when the random value is reset. The following three key names are significant: "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.

### 10.4.3 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:

\[
\{\text{value} \ <\text{operator}> \text{value}\}
\]

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

\[
\begin{align*}
\{\$[FOOBAR] * 0.5\} \\
\{-2-$[FOOBAR]\} \\
\{\$[APPLES] + $[PEARS]\} \\
\{35 / \{\$[FOOBAR]-2\}\} \\
\{\$[DBTIME] - \{35 / \{\$[UTCTIME]+2\}\}\}
\end{align*}
\]

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.

### 10.5 Random Time Warps and Initial 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.
10.5.1 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:

\[
\text{TIME\_WARP} = \langle\text{value}\rangle \\
\text{TIME\_WARP} = \langle\text{low\_value}\rangle:\langle\text{high\_value}\rangle
\]

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 10.7.2 for a use of random time warping to simulate random wind gusts.

10.5.2 Random Initial Start Delays

The start delay is 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 \(\text{elapsed\_time}=n\) at the outset of the script and all resets of the script. Thus a delay of \(n\) seconds combined with a time warp of 0.5 would result in observed delay of \(2\times n\) seconds. The start delay may be specified as a single value or a range of values as below:

\[
\text{START\_DELAY} = \langle\text{value}\rangle \\
\text{START\_DELAY} = \langle\text{low\_value}\rangle:\langle\text{high\_value}\rangle
\]

When a range of values is specified, the start delay value is calculated at the outset, and re-calculated whenever the script is reset. See the example in Section 10.7.1 for a use of random start delays to simulate the delay in acquiring satellite fixes in a GPS unit on an UUV coming to the surface.

10.6 More on uTimerScript Output to the MOOSDB and Console

The activity of uTimerScript may be monitored in two ways: (a) by a status message posted the MOOSDB, and by standard output to the uTimerScript console window.

10.6.1 Status Messages Posted to the MOOSDB by uTimerScript

The uTimerScript periodically publishes a string indicating the status of the script. The following is an example:

\[
\text{UTS\_STATUS} = \text{name=RND\_TEST, elapsed\_time=2.00, posted=1, pending=5, paused=false, conditions\_ok=true, time\_warp=3, start\_delay=0, shuffle=false, upon\_awake=restart, resets=2/5}
\]

In this case, the script has posted one of six events (\(\text{posted}=1, \text{pending}=5\)). It is actively unfolding, since \(\text{paused}=false\) (Section 10.3.1) and \(\text{conditions\_ok}=true\) (Section 10.3.2). It has been reset twice out of a maximum of five allowed resets (\(\text{resets}=2/5\), Section 10.2.2). Time warping is being
deployed (time_warp=3, Section 10.5.1), there is no start delay in use (start_delay=0, Section 10.5.2). The shuffle feature is turned off (shuffle=false, Section 10.2.2). The script is not configured to reset upon re-entering the un-paused state (awake_reset=false, Section 10.2.2).

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:

\[
\text{STATUS_VAR} = \text{<moos-variable>} \quad \text{// Default is UTS_STATUS}
\]

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.

\[
\text{SCRIPT_NAME} = \text{<string>} \quad \text{// Default is "unnamed"}
\]

10.6.2 Console Output Generated by uTimerScript

The script configuration and progress of script execution may also be monitored from an open console window where uTimerScript is launched, if the verbose setting is turned on (by default). Example output is shown below in Listing 37.

Listing 37 - Example uTimerScript console output.

0 Random Variable configurations:
1   [0]: varname=ANGLE, keyname=at_reset, min=10, max=350
2   [1]: varname=MAGN, keyname=at_reset, min=0.5, max=1.5
3
4 The Raw Script: ========================================
5 Total Elements: 10
6   [0] USM_FORCE_ANGLE=$[ANGLE], TIME:-1, RANGE=[0,0], POSTED=false
7   [1] USM_FORCE_MAGNITUDE_AAD={$[MAGN]*0.2}, TIME:-1, RANGE=[2,2], POSTED=false
8   [2] USM_FORCE_MAGNITUDE_AAD={$[MAGN]*0.2}, TIME:-1, RANGE=[4,4], POSTED=false
9   [3] USM_FORCE_MAGNITUDE_AAD={$[MAGN]*0.2}, TIME:-1, RANGE=[6,6], POSTED=false
10  [4] USM_FORCE_MAGNITUDE_AAD={$[MAGN]*0.2}, TIME:-1, RANGE=[8,8], POSTED=false
11  [5] USM_FORCE_MAGNITUDE_AAD={$[MAGN]*0.2}, TIME:-1, RANGE=[10,10], POSTED=false
12  [6] USM_FORCE_MAGNITUDE_AAD={$[MAGN]*-0.2}, TIME:-1, RANGE=[12,12], POSTED=false
13  [7] USM_FORCE_MAGNITUDE_AAD={$[MAGN]*-0.2}, TIME:-1, RANGE=[14,14], POSTED=false
14  [8] USM_FORCE_MAGNITUDE_AAD={$[MAGN]*-0.2}, TIME:-1, RANGE=[16,16], POSTED=false
15  [9] USM_FORCE_MAGNITUDE_AAD={$[MAGN]*-0.2}, TIME:-1, RANGE=[18,18], POSTED=false
16
17================================================================================
18
19 uTimerScript_wind is Running:
20    AppTick @ 5.0 Hz
21    CommsTick @ 5 Hz
22 Script (Re)Initialized. Time Warp=5 StartDelay=0
23  [a/239.512] [0]: USM_FORCE_ANGLE = 78.068
24  [u/239.915] [2.01267]: USM_FORCE_MAGNITUDE_AAD = 0.19936
25  [o/240.318] [4.02825]: USM_FORCE_MAGNITUDE_AAD = 0.19936
26  [i/240.722] [6.04633]: USM_FORCE_MAGNITUDE_AAD = 0.19936

84
In the first block (lines 1-2), the configuration of random variables for use as macros is displayed. In the second block (lines 5-17), the raw script, prior to macro expansion or time-stamp allocation is displayed. In the third block (lines 22-32), events are printed as they occur. Each event shows two timestamps. The first, on the left, shows the approximate time relative to the MOOSDB start time (which is typical in MOOS log files). The second set of timestamps shown in the second column is the "elapsed time" since the start of the script (which may be affected by time warp, start delay, and pausing).

10.7 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.

10.7.1 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 18, 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 18: **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 38 gets the job done.

*Listing 38 - A uTimerScript configuration for simulating aspects of a GPS unit.*

```plaintext
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  }
```

86
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 $[\text{COUNT}]$ macro in the value component. See Section 10.4.1 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 due to the RESET_MAX=unlimited and RESET_TIME=end settings in lines 10 and 11. This script is used in the IvP Helm example simulation mission labeled "s4_delta" illustrating the PeriodicSurface helm behavior.

### 10.7.2 A Script as a Proxy for Simulating Random Wind Gusts

Simulating wind gusts, or in general, somewhat random external periodic forces 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 forces 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 19), 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 force in the x-y plane, by default zero. An external force may be specified in terms of a force vector, in absolute terms with the variable USM_FORCE_VECTOR, or in relative terms with the variables USM_FORCE_VECTOR_ADD.

![Simulated UUV Operation](image)

Figure 19: **Simulated Wind Gusts**: The uTimerScript application may be configured to post periodic sequences of external force values, used by the uSimMarine application to simulate wind gust effects on its simulated vehicle.

The script in Listing 39 makes use of the uSimMarine interface by posting periodic force vectors. It simulates a wind gust with a sequence of five posts to increase a force vector (lines 18-22), and complementary sequence of five posts to decrease the force vector (lines 24-28) for a net force of zero at the end of each script execution.

### Listing 39 - A uTimerScript configuration for simulating simple wind gusts.

```plaintext
0  //-------------------------------
```
The force angle is chosen randomly in the range of \([0, 359]\) by use of the random variable macro \(\text{[ANG]}\) defined on line 16. The peak magnitude of the force vector is chosen randomly in the range of \([0, 1.5]\) with the random variable macro \(\text{[MAG]}\) defined on line 17. Note that these two macros have their random values reset each time the script begins, by using the \text{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 \text{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 \text{DELAY_RESTART} parameter on line 11. Used in conjunction with the \text{TIME_WARP} parameter, the interval for possible observed delays between gusts is \([5, 240]\). The \text{RESET_TIME=end} parameter on line 10 is used to ensure that the script posts all force vectors to avoid any accumulated forces over time. The \text{RESET_MAX} parameter is set to "unlimited" to ensure the script runs indefinitely.