Computer Science and Artificial Intelligence Laboratory

Technical Report

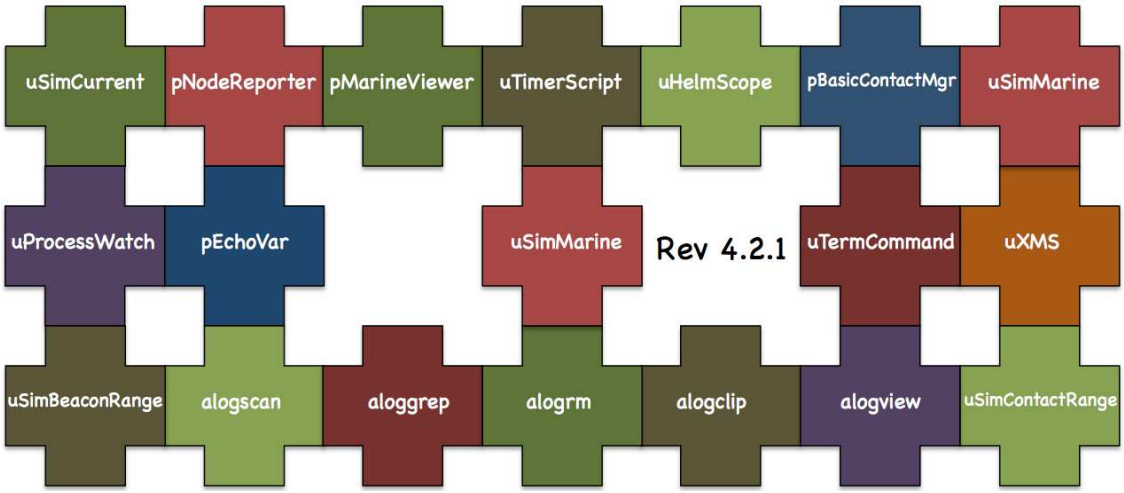
MIT-CSAIL-TR-2011-036 July 28, 2011

MOOS-IvP Autonomy Tools Users Manual Release 4.2.1

Michael R. Benjamin

massachusetts institute of technology, cambridge, ma 02139 usa — www.csail.mit.edu

MOOS-IvP Autonomy Tools Users Manual
Release 4.2.1



Michael R. Benjamin
Department Mechanical Engineering
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology, Cambridge MA

July 26, 2011 - Release 4.2.1

Abstract

This document describes 19 MOOS-IvP autonomy tools. *uHelmScope* provides a run-time scoping window into the state of an active IvP Helm executing its mission. *pMarineViewer* is a geo-based GUI tool for rendering marine vehicles and geometric data in their operational area. *uXMS* is a terminal based tool for scoping on a MOOSDB process. *uTermCommand* is a terminal based tool for poking a MOOSDB with a set of MOOS file pre-defined variable-value pairs selectable with aliases from the command-line. *pEchoVar* provides a way of echoing a post to one MOOS variable with a new post having the same value to a different variable. *uProcessWatch* monitors the presence or absence of a set of MOOS processes and summarizes the collective status in a single MOOS variable. *uPokeDB* provides a way of poking the MOOSDB from the command line with one or more variable-value pairs without any pre-existing configuration of a MOOS file. *uTimerScript* will execute a pre-defined timed pausable script of poking variable-value pairs to a MOOSDB. *pNodeReporter* summarizes a platforms critical information into a single node report string for sharing beyond the vehicle. *pBasicContactMgr* provides a basic contact management service with the ability to generate range-dependent configurable alerts. *uSimMarine* provides a simple marine vehicle simulator. *uSimBeaconRange* and *uSimContactRange* provide further simulation for range-only sensors. The *Alog Toolbox* is a set of offline tools for analyzing and manipulating log files in the .alog format.

This work is the product of a multi-year collaboration between the Department of Mechanical Engineering and the Computer Science and Artificial Intelligence Laboratory (CSAIL) at the Massachusetts Institute of Technology in Cambridge Massachusetts, and the Oxford University Mobile Robotics Group.

Points of contact for collaborators:

Dr. Michael R. Benjamin
Department of Mechanical Engineering
Computer Science and Artificial Intelligence Laboratory
Massachusetts Intitute of Technology
mikerb@csail.mit.edu

Prof. John J. Leonard
Department of Mechanical Engineering
Computer Science and Artificial Intelligence Laboratory
Massachusetts Intitute of Technology
jleonard@csail.mit.edu

Prof. Henrik Schmidt
Department of Mechanical Engineering
Massachusetts Intitute of Technology
henrik@mit.edu

Dr. Paul Newman
Department of Engineering Science
University of Oxford
pnewman@robots.ox.ac.uk

Other collaborators have contributed greatly to the development and testing of software and ideas within, notably - Joseph Curcio, Toby Schneider, Stephanie Kemna, Arjan Vermeij, Don Eickstedt, Andrew Patrikilakis, Arjuna Balasuriya, David Battle, Christian Convey, Chris Gagner, Andrew Shafer, and Kevin Cockrell.

Sponsorship, and public release information:

This work is sponsored by Dr. Behzad Kamgar-Parsi and Dr. Don Wagner of the Office of Naval Research (ONR), Code 311. Further support for testing and coursework development sponsored by Battelle, Dr. Robert Carnes.

Contents

1 Overview

1.1 Purpose and Scope of this Document

The MOOS-IvP autonomy tools described in this document are software applications that are typically running either as part of an overall autonomy system running on a marine vehicle, as part of a marine vehicle simulation. or used for post-mission off-line analysis. They are each MOOS applications, meaning they are running and communicating with a MOOSDB application as depicted in Figure 1. The AlogToolbox described here contains a number of off-line tools for analyzing alog files produced by the pLogger application.

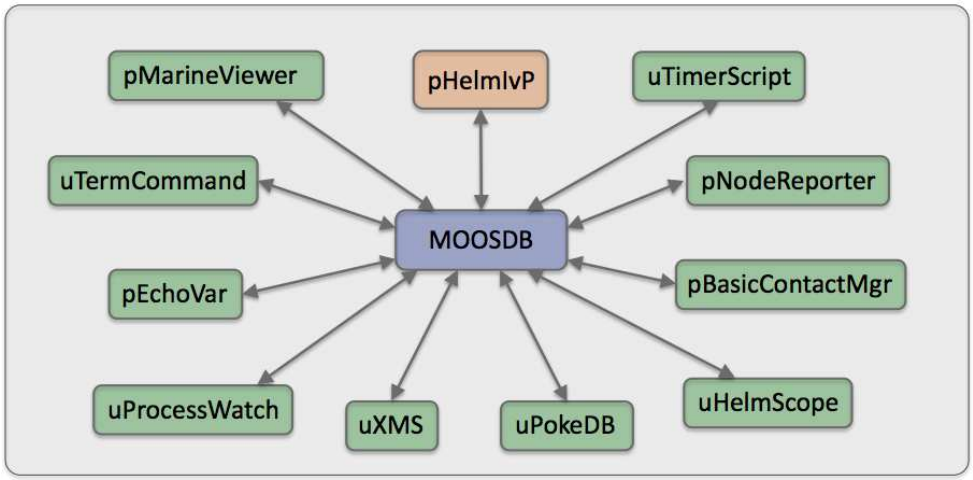


Figure 1: **Autonomy utility applications:** A MOOS community consists of a running MOOSDB application and a number of applications connected and communicating with each other via publish and subscribe interface. The pHelmIvP application provides the autonomy decision-making capapability and the other highlighted applications provide support capabilities for the system.

The focus of this paper is on these tools, and the set of off-line mission analysis tools comprising the Alog Toolbox. Important topics outside this scope are (a) MOOS middleware programming, (b) the IvP Helm and autonomy behaviors, and (c) other important MOOS utilities applications not covered here. The intention of this paper is to provide documenation for these common applications for current users of the MOOS-IvP software.

1.2 Brief Background of MOOS-IvP

MOOS was written by Paul Newman in 2001 to support operations with autonomous marine vehicles in the MIT Ocean Engineering and the MIT Sea Grant programs. At the time Newman was a post-doc working with John Leonard and has since joined the faculty of the Mobile Robotics Group at Oxford University. MOOS continues to be developed and maintained by Newman at Oxford and the most current version can be found at his web site. The MOOS software available in the MOOS-IvP project includes a snapshot of the MOOS code distributed from Oxford. The IvP Helm was developed in 2004 for autonomous control on unmanned marine surface craft, and later underwater platforms. It was written by Mike Benjamin as a post-doc working with John Leonard,

and as a research scientist for the Naval Undersea Warfare Center in Newport Rhode Island. The IvP Helm is a single MOOS process that uses multi-objective optimization to implement behavior coordination.

Acronyms

MOOS stands for "Mission Oriented Operating Suite" and its original use was for the Bluefin Odyssey III vehicle owned by MIT. IvP stands for "Interval Programming" which is a mathematical programming model for multi-objective optimization. In the IvP model each objective function is a piecewise linear construct where each piece is an *interval* in N-Space. The IvP model and algorithms are included in the IvP Helm software as the method for representing and reconciling the output of helm behaviors. The term interval programming was inspired by the mathematical programming models of linear programming (LP) and integer programming (IP). The pseudo-acronym IvP was chosen simply in this spirit and to avoid acronym clashing.

1.3 Sponsors of MOOS-IvP

Original development of MOOS and IvP were more or less infrastructure by-products of other sponsored research in (mostly marine) robotics. Those sponsors were primarily The Office of Naval Research (ONR), as well as the National Oceanic and Atmospheric Administration (NOAA). MOOS and IvP are currently funded by Code 31 at ONR, Dr. Don Wagner and Dr. Behzad Kamgar-Parsi. Testing and development of course work at MIT is further supported by Battelle, Dr. Robert Carnes. MOOS is additionally supported in the U.K. by EPSRC. Early development of IvP benefited from the support of the In-house Laboratory Independent Research (ILIR) program at the Naval Undersea Warfare Center in Newport RI. The ILIR program is funded by ONR.

1.4 The Software

The MOOS-IvP autonomy software is available at the following URL:

```
http://www.moos-ivp.org
```

Follow the links to *Software*. Instructions are provided for downloading the software from an SVN server with anonymous read-only access.

1.4.1 Building and Running the Software

This document is written to Release 4.2.1. After checking out the tree from the SVN server as prescribed at this link, the top level directory should have the following structure:

```
moos-ivp/
   MOOS@/
   MOOS-2374-Apr0611/
   README.txt
   README-LINUX.txt
   README-OS-X.txt
   README-WINDOWS.txt
   README.txt
```

```
bin/
build/
build-moos.sh
build-ivp.sh
configure-ivp.sh
ivp/
lib/
scripts/
```

Note there is a `MOOS` directory and an `IvP` sub-directory. The `MOOS` directory is a symbolic link to a particular MOOS revision checked out from the Oxford server. In the example above this is Revision 2374 on the Oxford SVN server. This directory is left completely untouched other than giving it the local name `MOOS-2374-Apr0611`. The use of a symbolic link is done to simplify the process of bringing in a new snapshot from the Oxford server.

The build instructions are maintained in the README files and are probably more up to date than this document can hope to remain. In short building the software amounts to two steps - building MOOS and building IvP. Building MOOS is done by executing the build-moos.sh script:

```
> cd moos-ivp
> ./build-moos.sh
```

Alternatively one can go directly into the `MOOS` directory and configure options with `ccmake` and build with `cmake`. The script is included to facilitate configuration of options to suit local use. Likewise the IvP directory can be built by executing the `build-ivp.sh` script. The `MOOS` tree must be built before building IvP. Once both trees have been built, the user's shell executable path must be augmented to include the two directories containing the new executables:

```
moos-ivp/MOOS/MOOSBin
moos-ivp/bin
```

At this point the software should be ready to run and a good way to confirm this is to run the example simulated mission in the missions directory:

```
> cd moos-ivp/ivp/missions/alpha/
> pAntler alpha.moos
```

Running the above should bring up a GUI with a simulated vehicle rendered. Clicking the `DEPLOY` button should start the vehicle on its mission. If this is not the case, some help and email contact links can be found at `www.moos-ivp.org/support/`, or emailing `issues@moos-ivp.org`.

1.4.2 Operating Systems Supported by MOOS and IvP

The MOOS software distributed by Oxford is well supported on Linux, Windows and Mac OS X. The software distributed by MIT includes additional MOOS utility applications and the IvP Helm and related behaviors. These modules are support on Linux and Mac OS X and the software compiles and runs on Windows but Windows support is limited.

1.5 Where to Get Further Information

1.5.1 Websites and Email Lists

There are two web sites - the MOOS web site maintained by Oxford University, and the MOOS-IvP web site maintained by MIT. At the time of this writing they are at the following URLs:

`http://www.robots.ox.ac.uk/~pnewman/TheMOOS/`

`http://www.moos-ivp.org`

What is the difference in content between the two web sites? As discussed previously, MOOS-IvP, as a set of software, refers to the software maintained and distributed from Oxford *plus* additional MOOS applications including the IvP Helm and library of behaviors. The software bundle released at moos-ivp.org does include the MOOS software from Oxford - usually a particular released version. For the absolute latest in the core MOOS software and documentation on Oxford MOOS modules, the Oxford web site is your source. For the latest on the core IvP Helm, behaviors, and MOOS tools distributed by MIT, the moos-ivp.org web site is the source.

There are two mailing lists open to the public. The first list is for MOOS users, and the second is for MOOS-IvP users. If the topic is related to one of the MOOS modules distributed from the Oxford web site, the proper email list is the "moosusers" mailing list. You can join the "moosusers" mailing list at the following URL:

`https://lists.csail.mit.edu/mailman/listinfo/moosusers,`

For topics related to the IvP Helm or modules distributed on the moos-ivp.org web site that are not part of the Oxford MOOS distribution (see the software page on moos-ivp.org for help in drawing the distinction), the "moosivp" mailing list is appropriate. You can join the "moosivp" mailing list at the following URL:

`https://lists.csail.mit.edu/mailman/listinfo/moosivp,`

1.5.2 Documentation

Documentation on MOOS can be found on the Oxford University web site:

`http://www.robots.ox.ac.uk/~pnewman/MOOSDocumentation/index.htm`

This includes documentation on the MOOS architecture, programming new MOOS applications as well as documentation on several bread-and-butter applications such as `pAntler`, `pLogger`, `uMS`, `pMOOSBridge`, `iRemote`, `iMatlab`, `pScheduler` and more. Documentation on the IvP Helm, behaviors and autonomy related MOOS applications not from Oxford can be found on the www.moos-ivp.org web site under the Documentation link. Below is a summary of documents:

List of available MOOS-IvP related documentation

- *An Overview of MOOS-IvP and a Brief Users Guide to the IvP Helm Autonomy Software* - This is the primary document describing the IvP Helm regarding how it works, the motivation for its design, how it is used and configured, and example configurations and results from simulation.
- *MOOS-IvP Autonomy Tools Users Manual* (this document) - A users manual for several MOOS applications, and off-line tools for post-mission analysis collectively referred to as the Alog Toolbox. The MOOS applications include: `pNodeReporter`, `uTimerScript`, `uHelmScope`, `uPokeDB`, `uSimMarine`, `uSimBeaconRange`, `uSimContactRange`, `pBasicContactMgr`, `uXMS`, `uTermCommand`, `pMarineViewer`, `pEchoVar`, and `uProcessWatch`. These applications are common supplementary tools for running an autonomy system in simulation and on the water.
- *Extending a MOOS-IvP Autonomy System and Users Guide to the IvPBuild Toolbox* - This document is a users manual for those wishing to write their own IvP Helm behaviors and MOOS modules. It describes the IvPBehavior and CMOOSApp superclass. It also describes the IvPBuild Toolbox containing a number of tools for building IvP Functions, the primary output of behaviors. It provides an example template directory with example IvP Helm behavior and an example MOOS application along with an example CMake build structure for linking against the standard software MOOS-IvP software bundle. MIT CSAIL Technical Report TR-2009-037.

1.6 What's New in Release 4.2.1 and 4.2

Below is a brief, likely incomplete, summary of changes and fixes notable in Release 4.2/4.2.1 since Release 4.1. The only difference between 4.2.1 and 4.2 is that the `uSimContactRange` app was renamed from the `uSimActiveSonar` app as it was known in 4.2, to avoid a name clash with a separately developed app by the same name with similar functionality.

pHelmIvP

- Improved support for initializing variables. Users have option of specifying whether variable initialization should override prevailing values in the MOOSDB or not.
- Journaling of IVPHELM_SUMMARY status reports to reduce log footprint
- Added support for the `--example`, `-e` example MOOS block cmdline switch
- Significant under-the-hood changes to allow for behaviors to produce multiple objective functions each.

alogview

- Many bug-fixes, drawing collective objective functions, scaling etc.
- Ability to view 1D (Depth) objective functions.
- Improvements under the hood in preparing for IvPBehaviors producing multiple objective functions each. Major change to the helm.
- Added support for rendering XYVector objects
- Added support for rendering XYRangePulse objects

- Added support for rendering XYMarker objects
- Handles rare case of empty logfiles - bug fix.

uSimMarine

- Revamped code formerly known as iMarineSim
- Added Speed-Over-Ground and Heading-Over-Ground
- Added support for publishing both a ground-truth and degraded navigation solution. Handing for testing navigation algorithms.
- Better support for external force vectors. Supports uSimCurrent
- Documentation added to the MOOS-IvP Autonomy Tools User Manual
- Added support for the `--example`, `-e` example MOOS block cmdline switch

pMarineViewer

- Stability fixes. A couple bugs caused crashes on Ubuntu systems.
- Added support for rendering XYVector objects
- Added support for rendering XYRangePulse objects
- Better support for determining the size of the OpGrid rendered
- Fixed Datum setting from the MOOS file rather than from the image .info file
- Allows for clearing of historical data

pNodeReporter

- Added support for publishing dual node reports when the simulator is publishing both a ground-truth and degraded navigation solution.
- Added support for the `--example`, `-e` example MOOS block cmdline switch

pBasicContactMgr

- - Minor fix to ignore reports with name matching ownship name.
- Added support for the `--example`, `-e` example MOOS block cmdline switch

uHelmScope

- Support for the new IVPHELM_SUMMARY journaling output of the helm
- Color output tied to change in helm decisions.
- Added support for the `--example`, `-e` example MOOS block cmdline switch

uTimerScript

- Added support for the `--example`, `-e` example MOOS block cmdline switch

uFunctionVis

- Many bug fixes especially in viewing collective objective functions

- Added support for view 1D (depth) objective functions

uSimCurrent

- A new application for simulating water current, coordinated with uSimMarine via uSimMarine's FORCE_VECTOR interface.

uSimContactRange

- A new application for simulating an on-board sensor that provides range measurements to other moving contacts.

uSimBeaconRange

- A new application for simulating an on-board sensor that provides a range measurement to a beacon where either (a) the vehicle knows where it is but is trying to determine the position of the beacon via a series of range measurements, or (b) the vehicle does not know where it is but is trying to determine its own position based on the range measurements from one or more beacons at known locations.

BHV_StationKeep

- Improved robustness in low-power mode in detecting zero progress recovering to station point.

2 The uHelmScope Utility: Scoping on the IvP Helm

2.1 Brief Overview

The `uHelmScope` application is a console based tool for monitoring output of the IvP helm, i.e., the `pHelmIvP` process. The helm produces a few key MOOS variables on each iteration that pack in a substantial amount of information about what happened during a particular iteration. The helm scope subscribes for and parses this information, and writes it to standard output in a console window for the user to monitor. The user can dynamically pause or alter the output format to suit one's needs, and multiple scopes can be run simultaneously. The helm scope in no way influences the performance of the helm - it is strictly a passive observer.

2.2 Console Output of `uHelmScope`

The example console output shown in Listing 1 is used for explaining the `uHelmScope` fields.

Listing 1 - Example `uHelmScope` *output.*

```
 1  ==============   uHelmScope Report  ============== ENGAGED  (17)
 2    Helm Iteration: 66      (hz=0.38)(5)  (hz=0.35)(66)  (hz=0.56)(max)
 3    IvP functions:  1
 4    Mode(s):        Surveying
 5    SolveTime:      0.00    (max=0.00)
 6    CreateTime:     0.02    (max=0.02)
 7    LoopTime:       0.02    (max=0.02)
 8    Halted:         false   (0 warnings)
 9  Helm Decision: [speed,0,4,21] [course,0,359,360]
10    speed = 3.00
11    course = 177.00
12  Behaviors Active: ---------- (1)
13    waypt_survey (13.0) (pwt=100.00) (pcs=1227) (cpu=0.01) (upd=0/0)
14  Behaviors Running: --------- (0)
15  Behaviors Idle: ------------ (1)
16    waypt_return (22.8)
17  Behaviors Completed: ------- (0)
18
19  #  MOOSDB-SCOPE ------------------------------------ (Hit '#' to en/disable)
20  #
21  #  VarName           Source       Time     Community  VarValue
22  #  ----------------  -----------  -------  ---------  -----------
23  #  BHV_WARNING        n/a          n/a      n/a       n/a
24  #  NODE_REPORT_LOCAL pNode..rter  24.32    alpha      "NAME=alpha,TYPE=KAYAK,MOOSDB"+
25  #  DEPLOY*           iRemote      11.25    alpha      "true"
26  #  RETURN*           pHelmIvP     5.21     alpha      "false"
27
28  @  BEHAVIOR-POSTS TO MOOSDB ---------------------- (Hit '@' to en/disable)
29  @
30  @  MOOS Variable     Value
31  @  -------------     -------  (BEHAVIOR=waypt_survey)
32  @  PC_waypt_survey   -- ok --
33  @  WPT_STAT_LOCAL    vname=alpha,index=1,dist=80.47698,eta=26.83870
34  @  WPT_INDEX         1
35  @  VIEW_SEGLIST      label,alpha_waypt_survey : 30,-20:30,-100:90,-100: +
36  @  -------------     -------  (BEHAVIOR=waypt_return)
37  @  PC_waypt_return   RETURN = true
38  @  VIEW_SEGLIST      label,alpha_waypt_return : 0,0
39  @  VIEW_POINT        0,0,0,waypt_return
```

There are three groups of information in the `uHelmScope` output on each report to the console - the general helm overview (lines 1-17), a `MOOSDB` scope for a select subset of MOOS variables (lines

19-26), and a report on the MOOS variables published by the helm on the current iteration (lines 28-39). The output of each group is explained in the next three subsections.

2.2.1 The General Helm Overview Section of the `uHelmScope` Output

The first block of output produced by `uHelmScope` provides an overview of the helm. This is lines 1-17 in Listing 1, but the number of lines may vary with the mission and state of mission execution. The integer value at the end of line 1 indicates the number of `uHelmScope` reports written to the console. This can confirm to the user that an action that should result in a new report generation has indeed worked properly. The integer on line 2 is the counter kept by the helm, incremented on each helm iteration. The three sets of numbers that follow indicate the observed time between helm iterations. These numbers are reported by the helm and are not inferred by the scope. The first number is the average over the most recent five iterations. The second is the average over the most recent 58 iterations. The last is the maximum helm-reported interval observed by the scope. The number of iterations used to generate the first two numbers can be set by the user in the `uHelmScope` configuration block. The default is 5 and 100 respectively. The number 58 is shown in the second group simply because 100 iterations hadn't been observed yet. The helm is apparently only on iteration 66 in this example and `uHelmScope` apparently didn't start and connect to the `MOOSDB` until the helm was on iteration 8.

The value on Line 3 represents the the number of IvP functions produced by the active helm behaviors, one per active behavior. The solve-time on line 5 represents the time, in seconds, needed to solve the IvP problem comprised the n IvP functions. The number that follows in parentheses is the maximum solve-time observed by the scope. The create-time on line 6 is the total time needed by all active behaviors to produce their IvP function output. The loop time on line 7 is simply the sum of lines 5 and 6. The Boolean on line 8 is true only if the helm is halted on an emergency or critical error condition. Also on line 8 is the number of warnings generated by the helm. This number is reported by the helm and *not* simply the number of warnings observed by the scope. This number coincides with the number of times the helm writes a new message to the variable `BHV_WARNING`.

The helm decision space (i.e., IvP domain) is displayed on line 9, with the following lines used to display the actual helm decision. Following this is a list of all the active, running, idle and completed behaviors. At any point in time, each instantiated IvP behavior is in one of these four states and each behavior specified in the behavior file should appear in one of these groups. Technically all *active* behaviors are also *running* behaviors but not vice versa. So only the running behaviors that are not active (i.e., the behaviors that could have, but chose not to produce an objective function), are listed in the "`Behaviors Running:`" group. Immediately following each behavior the time, in seconds, that the behavior has been in the current state is shown in parentheses. For the active behaviors (see line 13) this information is followed by the priority weight of the behavior, the number of pieces in the produced IvP function, and the amount of CPU time required to build the function. If the behavior also is accepting dynamic parameter updates the last piece of information on line 13 shows how many successful updates where made against how many attempts. A failed update attempt also generates a helm warning, counted on line 8. The idle and completed behaviors are listed by default one per line. This can be changed to list them on one long line by hitting the 'b' key interactively. Insight into why an idle behavior is not in the running state can be found in the another part of the report (e.g., line 37) described below in Section 2.2.3.

2.2.2 The MOOSDB-Scope Section of the uHelmScope Output

Part of understanding what is happening in the helm involves the monitoring of variables in the MOOSDB that can either affect the helm or reveal what is being produced by the helm. Although there are other MOOS scope tools available (e.g., uXMS or uMS), this feature does two things the other scopes do not. First, it is simply a convenience for the user to monitor a few key variables in the same screen space. Second, uHelmScope automatically registers for the variables that the helm reasons over to determine the behavior activity states. It will register for all variables appearing in behavior conditions, runflags, activeflags, inactiveflags, endflags and idleflags. Variables that are registered for by this criteria are indicated by an asterisk at the end of the variable name. If the output resulting from these automatic registrations becomes unwanted, it can be toggled off by typing 's'.

The lines comprising the MOOSDB-Scope section of the uHelmScope output are all preceded by the '#' character. This is to help discern this block from the others, and as a reminder that the whole block can be toggled off and on by typing the '#' character. The columns in Listing 1 are truncated to a set maximum width for readability. The default is to have truncation turned off. The mode can be toggled by the console user with the 't' character, or set in the MOOS configuration block or with a command line switch. A truncated entry in the VarValue column has a '+' at the end of the line. Truncated entries in other columns will have ".." embedded in the entry. Line 24 shows an example of both kinds of truncation.

The variables included in the scope list can be specified in the uHelmScope configuration block of a MOOS file. In the MOOS file, the lines have the form:

```
VAR = VARIABLE_1, VARIABLE_2, VARIABLE_3, ...
```

An example configuration is given in Listing 4. Variables can also be given on the command line. Duplicates requests, should they occur, are simply ignored. Occasionally a console user may want to suppress the scoping of variables listed in the MOOS file and instead only scope on a couple variables given on the command line. The command line switch -c will suppress the variables listed in the MOOS file - unless a variable is also given on the command line. In line 23 of Listing 1, the variable BHV_WARNING is a *virgin* variable, i.e., it has yet to be written to by any MOOS process and shows n/a in the four output columns. By default, virgin variables are displayed, but their display can be toggled by the console user by typing '-v'.

2.2.3 The Behavior-Posts Section of the uHelmScope Output

The Behavior-Posts section is the third group of output in uHelmScope lists MOOS variables and values posted by the helm on the current iteration. Each variable was posted by a particular helm behavior and the grouping in the output is accordingly by behavior. Unlike the variables in the MOOSDB-Scope section, entries in this section only appear if they were written to on the current iteration. The lines comprising the Behavior-Posts section of the uHelmScope output are all preceded by the '@' character. This is to help discern this block from the others, and as a reminder that the whole block can be toggled off and on by typing the '@' character. As with the output in the MOOSDB-Scope output section, the output may be truncated. A trailing '+' at the end of the line indicates the variable value has been truncated.

There are a few switches for keeping the output in this section concise. A behavior posts a few standard MOOS variables on every iteration that may be essentially clutter for users in most cases. A behavior `FOO` for example produces the variables `PWT_FOO`, `STATE_FOO`, and `UH_FOO` which indicate the priority weight, run-state, and tally of successful updates respectively. Since this information is present in other parts of the `uHelmScope` output, these variables are by default suppressed in the Behavior-Posts output. Two other standard variables are `PC_FOO` and `VIEW_*` which indicate the precondition keeping a behavior in an idle state, and standard viewing hints to a rendering engine. Since this information is not present elsewhere in the `uHelmScope` output, it is not masked out by default. A console user can mask out the `PWT`, `STATE_*` and `UH_*` variables by typing 'm'. The `PC_*` and `VIEW_*` variables can be masked out by typing 'M'. All masked variables can be unmasked by typing 'u'.

2.3 Stepping Forward and Backward Through Saved Scope History

The user has the option of pausing and stepping forward or backward through helm iterations to analyze how a set of events may have unfolded. Stepping one event forward or backward can be done with the '[' and ']' keys respectively. Stepping 10 or 100 events can be done with the '{' and '}, and '(' and ')' keys respectively. The current helm iteration being displayed is always shown on the second line of the output. For each helm iteration, the `uHelmScope` process stores the information published by the helm (Section 2.5), and thus the memory usage of `uHelmScope` would grow unbounded if left unchecked. Therefore information is kept for a maximum of 2000 helm iterations. This number is *not* a configuration parameter - to preclude a user from inadvertently setting this too high and inducing the system maladies of a single process with runaway memory usage. To change this number, a user must change the source code (in particular the variable `m_history_size_max` in the file HelmScope.cpp). The `uHelmScope` history is therefore a moving window of fixed size that continues to shift right as new helm information is received. Stepping forward or backwards therefore is subject to the constraints of this window. Any steps backward or forward will in effect generate a new *requested* helm index for viewing. The requested index, if older than the oldest stored index, will be set exactly to the oldest stored index. Similarly in the other direction. It's quite possible then to hit the '[' key to step left by one index, and have the result be a report that is not one index older, but rather some number of indexes newer. Hitting the space bar or 'r' key always generates a report for the very latest helm information, with the 'r' putting the scope into streaming, i.e., continuous update, mode.

2.4 Console Key Mapping and Command Line Usage Summaries

The `uHelmScope` has a separate thread to accept user input from the console to adjust the content and format of the console output. It operates in either the *streaming mode*, where new helm summaries are displayed as soon as they are received, or the *paused mode* where no further output is generated until the user requests it. The key mappings can be summarized in the console output by typing the 'h' key, which also sets the mode to *paused*. The key mappings shown to the user are shown in Listing 2.

Listing 2 - Key mapping summary shown after hitting 'h' in a console.

```
1  KeyStroke  Function
2  ---------  --------------------------
```

```
 3     Spc     Pause and Update latest information once - now
 4     r/R     Resume information refresh
 5     h/H     Show this Help msg - 'r' to resume
 6     b/B     Toggle Show Idle/Completed Behavior Details
 7     t/T     Toggle truncation of column output
 8     m/M     Toggle display of Hiearchical Mode Declarations
 9      f      Filter PWT_* UH_* STATE_* in Behavior-Posts Report
10      F      Filter PC_* VIEW_* in Behavior-Posts Report
11     s/S     Toggle Behavior State Vars in MOOSDB-Scope Report
12     u/U     Unmask all variables in Behavior-Posts Report
13     v/V     Toggle display of virgins in MOOSDB-Scope output
14     [/]     Display Iteration 1 step prev/forward
15     {/}     Display Iteration 10 steps prev/forward
16     (/)     Display Iteration 100 steps prev/forward
17      #      Toggle Show the MOOSDB-Scope Report
18      @      Toggle Show the Behavior-Posts Report
19
20  Hit 'r' to resume outputs, or SPACEBAR for a single update
```

Several of the same preferences for adjusting the content and format of the `uHelmScope` output can be expressed on the command line, with a command line switch. The switches available are shown to the user by typing `uHelmScope -h`. The output shown to the user is shown in Listing 3.

Listing 3 - Command line usage of the `uHelmScope` *application.*

```
 1  > uHelmScope -h
 2  Usage: uHelmScope moosfile.moos [switches] [MOOSVARS]
 3    -t:  Column truncation is on (off by default)
 4    -c:  Exclude MOOS Vars in MOOS file from MOOSDB-Scope
 5    -x:  Suppress MOOSDB-Scope output block
 6    -p:  Suppress Behavior-Posts output block
 7    -v:  Suppress display of virgins in MOOSDB-Scope block
 8    -r:  Streaming (unpaused) output of helm iterations
 9    MOOSVAR_1 MOOSVAR_2 .... MOOSVAR_N
```

The command line invocation also accepts any number of MOOS variables to be included in the MOOSDB-Scope portion of the `uHelmScope` output. Any argument on the command line that does not end in `.moos`, and is not one of the switches listed above, is interpreted to be a requested MOOS variable for inclusion in the scope list. Thus the order of the switches and MOOS variables do not matter. These variables are added to the list of variables that may have been specified in the `uHelmScope` configuration block of the MOOS file. Scoping on *only* the variables given on the command line can be accomplished using the `-c` switch. To support the simultaneous running of more than one `uHelmScope` connected to the same `MOOSDB`, `uHelmScope` generates a random number N between 0 and 10,000 and registers with the `MOOSDB` as `uHelmScope_N`.

2.5 IvPHelm MOOS Variable Output Supporting uHelmScope Reports

There are six variables published by the `pHelmIvP` MOOS process, and registered for by the `uHelmScope` process, that provide critical information for generating `uHelmScope` reports. They are: `IVPHELM_SUMMARY`, `IVPHELM_POSTINGS`, `IVPHELM_ENGAGED`, `IVPHELM_STATEVARS`, `IVPHELM_DOMAIN`, `IVPHELM_LIFE_EVENT` and `IVPHELM_MODESET`. The first three are produced on each iteration of the helm, and the last three are typically only produced once when the helm is launched.

```
IVPHELM_SUMMARY  = "iter=66,ofnum=1,warnings=0,utc_time=1209755370.74,solve_time=0.00,
     create_time=0.02,loop_time=0.02,var=speed:3.0,var=course:108.0,halted=false,
     running_bhvs=none,active_bhvs=waypt_survey$6.8$100.00$1236$0.01$0/0,
     modes=MODE@ACTIVE:SURVEYING,idle_bhvs=waypt_return$55.3$n/a,completed_bhvs=none"

IVPHELM_POSTINGS  = "waypt_return$@!$66$@!$PC_waypt_return=RETURN = true$@!$VIEW_SEGLIST=label,
     alpha_waypt_return : 0,0$@!$VIEW_POINT=0,0,0,waypt_return$@!$PWT_BHV_WAYPT_RETURN=0
     $@!$STATE_BHV_WAYPT_RETURN=0"

IVPHELM_POSTINGS  = waypt_survey$@!$66$@!$PC_waypt_survey=-- ok --$@!$WPT_STAT_LOCAL=vname=alpha,
     index=1,dist=80.47698,eta=26.83870$@!$WPT_INDEX=1$@!$VIEW_SEGLIST=label,
     alpha_waypt_survey:30,-20:30,-100:90,-100:110,-60:90,-20$@!$PWT_BHV_WAYPT_SURVEY=100$@!$
     STATE_BHV_WAYPT_SURVEY=2

IVPHELM_DOMAIN = "speed,0,4,21:course,0,359,360"

IVPHELM_STATEVARS = "RETURN,DEPLOY"

IVPHELM_MODESET   = "---,ACTIVE#---,INACTIVE#ACTIVE,SURVEYING#ACTIVE,RETURNING"

IVPHELM_ENGAGED   = "ENGAGED"
```

The `IVPHELM_SUMMARY` variable contains all the dynamic information included in the general helm overview (top) section of the uHelmScope output. It is a comma-separated list of `var=val` pairs. The `IVP_DOMAIN` variable also contributes to this section of output by providing the IvP domain used by the helm. The `IVPHELM_POSTINGS` variable includes a list of MOOS variables and values posted by the helm for a given behavior. The helm writes to this variable once per iteration *for each behavior*. The `IVPHELM_STATEVARS` variable affects the MOOSDB-Scope section of the `uHelmScope` output by identifying which MOOS variables are used by behaviors in conditions, runflags, endflags and idleflags.

2.6 Configuration Parameters for uHelmScope

Configuration for `uHelmScope` amounts to specifying a set of parameters affecting the terminal output format. An example configuration is shown in Listing 4, with all values set to the defaults. Launching `uHelmScope` with a MOOS file that does not contain a `uHelmScope` configuration block is perfectly reasonable. To see an example MOOS configuration block, enter the following from the command-line:

```
$ uHelmScope -e
```

This will show the output shown in Listing 4 below.

Listing 4 - Example configuration of the uHelmScope application.

```
0  ============================================================
1  uHelmScope Example MOOS Configuration
2  ============================================================
3  Blue lines:     Default configuration
4  Magenta lines:  Non-default configuration
5
```

```
 6  ProcessConfig = uHelmScope
 7  {
 8    AppTick   = 1    // MOOSApp default is 4
 9    CommsTick = 1    // MOOSApp default is 4
10
11    paused    = false
12
13    hz_memory = 5,100
14
15    display_moos_scope = true     // or {false}
16    display_bhv_posts  = true     // or {false}
17    display_virgins    = true     // or {false}
18    display_statevars  = true     // or {false}
19    truncated_output   = false    // or {true}
20    behaviors_concise  = true     // or {false}
21
22    var       = NAV_X, NAV_Y, NAV_SPEED, NAV_DEPTH
23    var       = DESIRED_HEADING, DESIRED_SPEED
24  }
```

Each of the parameters, with the exception of HZ_MEMORY can also be set on the command line, or interactively at the console, with one of the switches or keyboard mappings listed in Section 2.4. A parameter setting in the MOOS configuration block will take precedence over a command line switch. The HZ_MEMORY parameter takes two integer values, the second of which must be larger than the first. This is the number of samples used to form the average time between helm intervals, displayed on line 2 of the uHelmScope output.

2.7 Publications and Subscriptions for uHelmScope

Variables published by the uHelmScope application

- NONE

Variables subscribed for by the uHelmScope application

- <USER-DEFINED>: Variables identified for scoping by the user in the uHelmScope will be subscribed for. See Section 2.2.2.
- <HELM-DEFINED>: As described in Section 2.2.2, the variables scoped by uHelmScope include any variables involved in the preconditions, runflags, idleflags, activeflags, inactiveflags, and endflags for any of the behaviors involved in the current helm configuration.
- IVPHELM_SUMMARY: See Section 2.5.
- IVPHELM_POSTINGS: See Section 2.5.
- IVPHELM_STATEVARS: See Section 2.5.
- IVPHELM_DOMAIN: See Section 2.5.
- IVPHELM_MODESET: See Section 2.5.
- IVPHELM_ENGAGED: See Section 2.5.

3 The pMarineViewer Utility: A GUI for Mission Control

3.1 Brief Overview

The `pMarineViewer` application is a MOOS application written with FLTK and OpenGL for rendering vehicles and associated information and history during operation or simulation. The typical layout shown in Figure 2 is that `pMarineViewer` is running in its own dedicated local MOOS community while simulated or real vehicles on the water transmit information in the form of a stream of *node reports* to the local community.

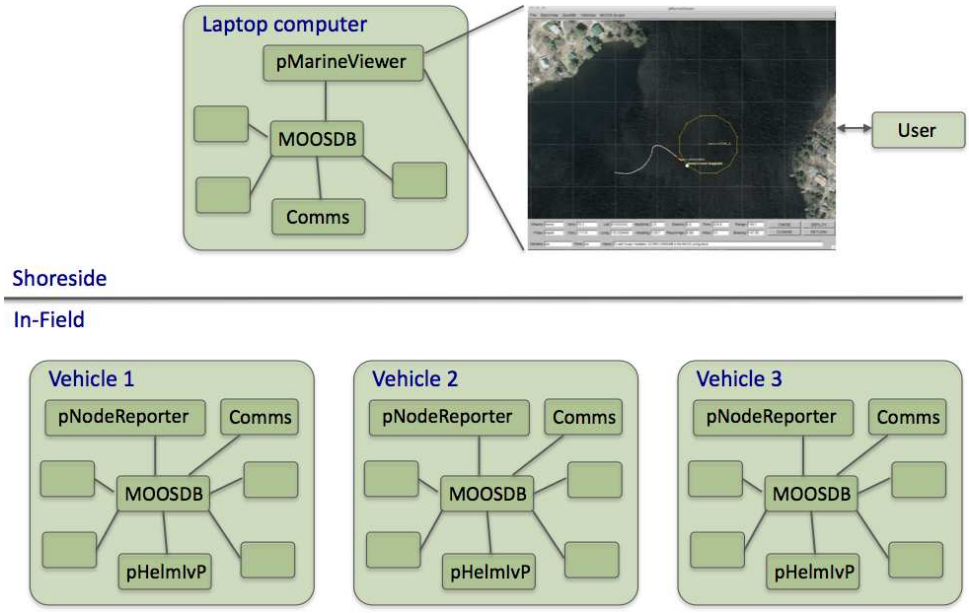


Figure 2: A common usage of the `pMarineViewer` is to have it running in a local `MOOSDB` community while receiving node reports on vehicle poise from other MOOS communities running on either real or simulated vehicles. The vehicles can also send messages with certain geometric information such as polygons and points that the view will accept and render.

The user is able manipulate a geo display to see multiple vehicle tracks and monitor key information about individual vehicles. In the primary interface mode the user is a passive observer, only able to manipulate what it sees and not able to initiate communications to the vehicles. However there are hooks available and described later in this section to allow the interface to accept field control commands.

A key variable subscribed to by `pMarineViewer` is the variable `NODE_REPORT`, which has the following structure given by an example:

```
NODE_REPORT = "NAME=nyak201,TYPE=kayak,UTC_TIME=1195844687.236,X=37.49,Y=-47.36,
              SPD=2.40,HDG=11.17,DEPTH=0"
```

Reports from different vehicles are sorted by their vehicle name and stored in histories locally in the `pMarineViewer` application. The `NODE_REPORT` is generated by the vehicles based on either sensor information, e.g., GPS or compass, or based on a local vehicle simulator.

3.2 Description of the pMarineViewer GUI Interface

The viewable area of the GUI has two parts - a geo display area where vehicles and perhaps other objects are rendered, and a lower area with certain data fields associated with an *active* vehicle are updated. A typical screen shot is shown in Figure 3 with two vehicles rendered - one AUV and one kayak. Vehicle labels and history are rendered. Properties of the vehicle rendering such as the trail length, size, and color, and vehicle size and color, and pan and zoom can be adjusted dynamically in the GUI. They can also be set in the `pMarineViewer` MOOS configuration block. Both methods of tuning the rendering parameters are described later in this section.

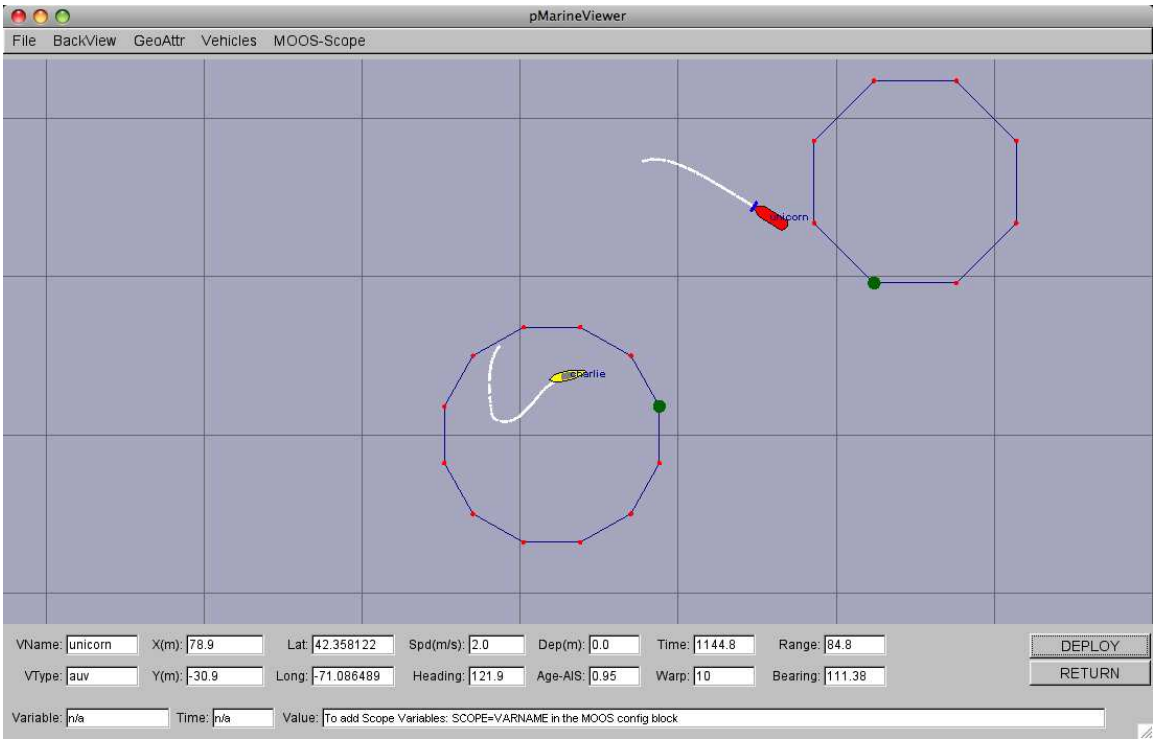


Figure 3: A screen shot of the `pMarineViewer` application running with two vehicles - one kayak platform, and one AUV platform. The Unicorn AUV platform is the *active* platform meaning the data fields on the bottom reflect the data for this platform.

The lower part of the display is dedicated to displaying detailed position information on a single *active* vehicle. Changing the designation of which vehicle is active can be accomplished by repeatedly hitting the 'v' key. The active vehicle is always rendered as red, while the non-active vehicles have a default color of yellow. Individual vehicle colors can be given different default values (even red, which could be confusing) by the user. The individual fields are described below:

- `VName:` The name of the active vehicle associated with the data in the other GUI data fields. The active vehicle is typically indicated also by changing to the color red on the geo display.
- `VType:` The platform type, e.g., AUV, Glider, Kayak, Ship or Unknown.
- `X(m):` The x (horizontal) position of the active vehicle given in meters in the local coordinate system.
- `Y(m):` The y (vertical) position of the active vehicle given in meters in the local coordinate system.

- **Lat:** The latitude (vertical) position of the active vehicle given in decimal latitude coordinates.
- **Lon:** The longitude (horizontal) position of the active vehicle given in decimal longitude coordinates.
- **Speed:** The speed of the active vehicle given in meters per second.
- **Heading:** The heading of the active vehicle given in degrees ($0 - 359.99$).
- **Depth:** The depth of the active vehicle given in meters.
- **Report-AGE:** The elapsed time in seconds since the last received node report for the active vehicle.
- **Time:** Time in seconds since the pMarineViewer process launched.
- **Warp:** The MOOS Time-Warp value. Simulations may run faster than real-time by this warp factor. MOOSTimeWarp is set as a global configuration parameter in the `.moos` file.
- **Range:** The range (in meters) of the active vehicle to a reference point. By default, this point is the datum, or the (0,0) point in local coordinates. The reference point may also be set to another particular vehicle. See Section 3.3.7 on the `ReferencePoint` pull-down menu.
- **Bearing:** The bearing (in degrees) of the active vehicle to a reference point. By default, this point is the datum, or the (0,0) point in local coordinates. The reference point may also be set to another particular vehicle. See Section 3.3.7 on the `ReferencePoint` pull-down menu.

In simulation, the age of the node report is likely to remain zero as shown in the figure, but when operating on the water, monitoring the node report age field can be the first indicator when a vehicle has failed or lost communications. Or it can act as an indicator of comms quality.

The lower three fields of the window are used for scoping on a single MOOS variable. See Section 3.3.4 for information on how to configure the `pMarineViewer` to scope on any number of MOOS variables and select a single variable via an optional pull-down menu. The scope fields are:

- **Variable:** The variable name of the MOOS variable currently being scoped, or `"n/a"` if no scope variables are configured.
- **Time:** The variable name of the MOOS variable currently being scoped, or `"n/a"` if no scope variables are configured.

3.3 Pull-Down Menu Options

Properties of the geo display rendering can be tuned to better suit a user or circumstance or for situations where screen shots are intended for use in other media such as papers or PowerPoint. There are two pull-down menus - the first deals with background properties, and the second deals with properties of the objects rendered on the foreground. Many of the adjustable properties can be adjusted by two other means besides the pull-down menus - by the hot keys defined for a particular pull-down menu item, or by configuring the parameter in the MOOS file configuration block.

3.3.1 The "BackView" Pull-Down Menu

Most pull-down menu items have hot keys defined (on the right in the menu). For certain actions like pan and zoom, in practice the typical user quickly adopts the hot-key interface. But the pull-down menu is one way to have a form of hot-key documentation always handy. The zooming commands affect the viewable area and apparent size of the objects. Zoom in with the 'i' or 'I' key, and zoom out with the 'o' or 'O' key. Return to the original zoom with ctrl+'z'.

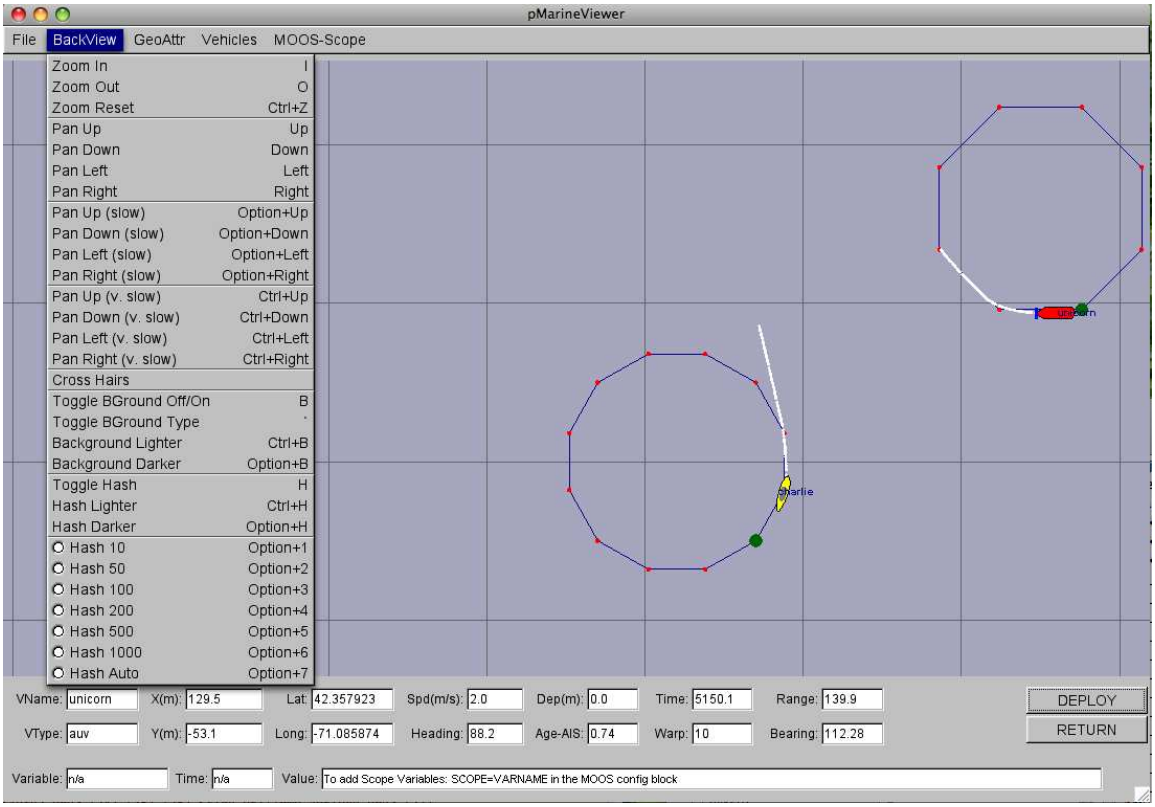


Figure 4: **The BackView menu:** This pull-down menu lists the options, with hot-keys, for affecting rendering aspects of the geo-display background.

Panning is done with the keyboard arrow keys. Three rates of panning are supported. To pan in 20 meter increments, just use the arrow keys. To pan "slowly" in one meter increments, use the Alt + arrow keys. And to pan "very slowly", in increments of a tenth of a meter, use the Ctrl + arrow keys. The viewer supports two types of "convenience" panning. It will pan put the active vehicle in the center of the screen with the 'C' key, and will pan to put the average of all vehicle positions at the center of the screen with the 'c' key. These are part of the 'Vehicles' pull-down menu discussed in Section 3.3.3.

The background can be in one of two modes; either displaying a gray-scale background, or displaying a geo image read in as a texture into OpenGL from an image file. The default is the geo display mode if provided on start up, or the grey-scale mode if no image is provided. The mode can be toggled by typing the 'b' or 'B' key. The geo-display mode can have two sub-modes if two image files are provided on start-up. More on this in Section 3.7. This is useful if the user has access to a satellite image *and* a map image for the same operation area. The two can be toggled by hitting the back tick key. When in the grey-scale mode, the background can be made lighter by hitting the ctrl+'b' key, and darker by hitting the alt+'b' key.

Hash marks can be overlaid onto the background. By default this mode is off, but can be toggled with the 'h' or 'H' key. The hash marks are drawn in a grey-scale which can be made lighter by typing the ctrl+'h' key, and darker by typing the alt+'h' key. Certain hash parameters can also be set in the `pMarineViewer` configuration block of the MOOS file. The `hash_view` parameter can

be set to either `true` or `false`. The default is `false`. The `hash_delta` parameter can be set to any integer in the range $[10, 1000]$. The default is 100.

3.3.2 The "GeoAttributes" Pull-Down Menu

The GeoAttributes pull-down menu allows the user to edit the properties of geometric objects capable of being rendered by the `pMarineViewer`. In general the Polygon, SegList, Point, and XYGrid objects are received by the viewer at run time to reflect artifacts generated by the IvP Helm indicating aspects of progress during their mission. The polygons in Figure 5 for example represents the set of waypoints being used by the vehicles shown.

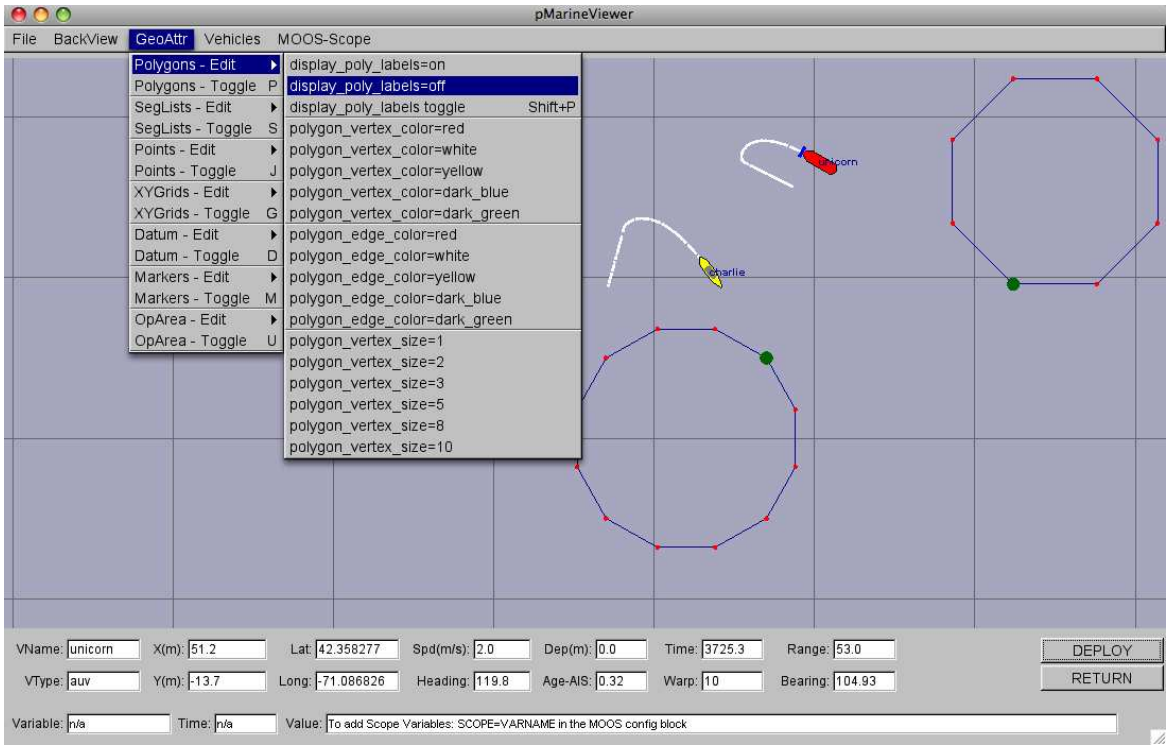


Figure 5: **The GeoAttributes menu:** This pull-down menu lists the options and hot keys for affecting the rendering of geometric objects.

The Datum, Marker and OpArea objects are typically read in once at start-up and reflect persistent info about the operation area. The datum is a single point that represents (0,0) in local coordinates. Marker objects typically represent physical objects in the environment such as a buoy, or a fixed sensor. The OpArea objects are typically a combination of points and lines that reflect a region of earth where a set of vehicles are being operated. Each category has a hot key that toggles the rendering of all objects of the same type, and a secondary drop-down menu as shown in the figure that allows the adjustment of certain rendering properties of objects. Many of the items in the menu have form `parameter = value`, and these settings can also be achieved by including this line in the `pMarineViewer` configuration block in the MOOS file.

3.3.3 The "Vehicles" Pull-Down Menu

The *Vehicles* pull-down menu deals with properties of the objects displayed in the geo display foreground. The `Vehicles-Toggle` menu item will toggle the rendering of all vehicles and all trails. The `Cycle_Focus` menu item will set the index of the *active* vehicle, i.e., the vehicle who's attributes are being displayed in the lower output boxes. The assignment of an index to a vehicle depends on the arrival of node reports. If an node report arrives for a previously unknown vehicle, it is assigned a new index.

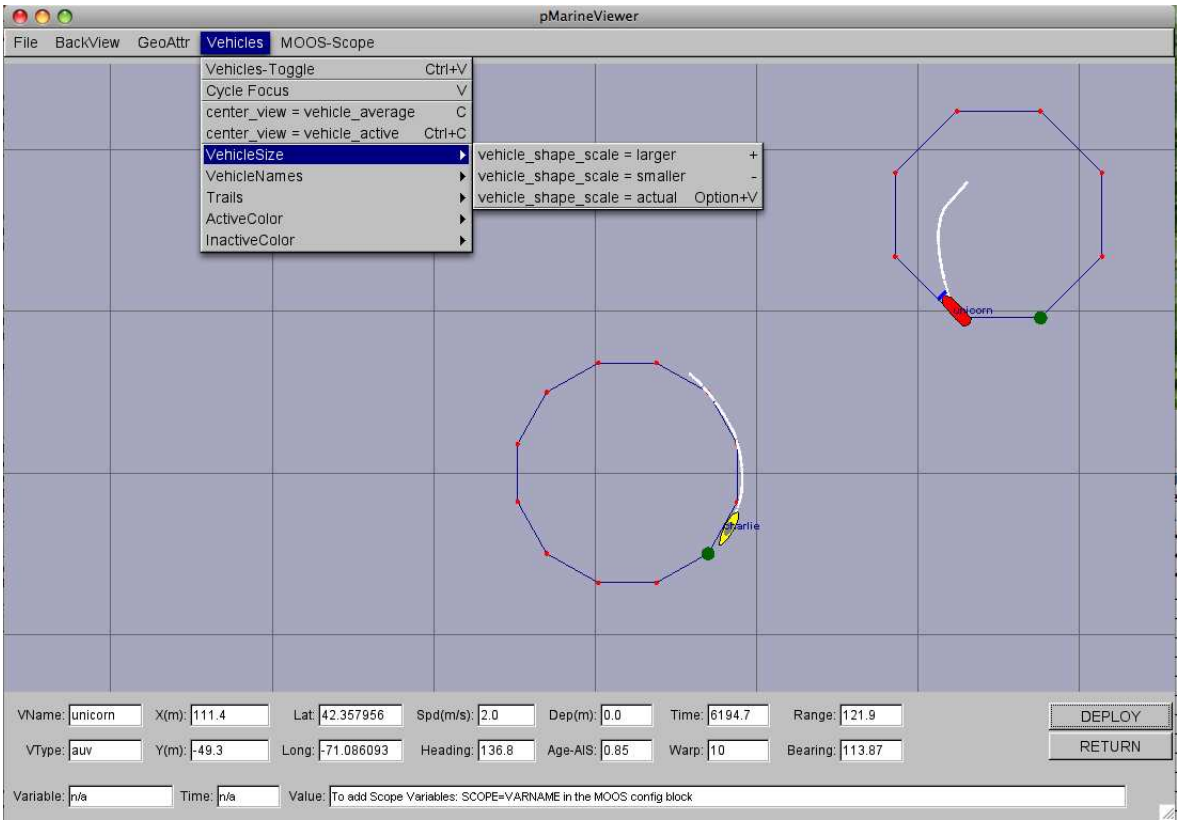


Figure 6: **The ForeView menu:** this pull-down menu of the `pMarineViewer` lists the options, with hot-keys, for affecting rendering aspects of the objects on the geo-display foreground, such as vehicles and vehicle track history.

The `center_view` menu items alters the center of the view screen to be panned to either the position of the active vehicle, or the position representing the average of all vehicle positions. Once the user has selected this, this mode remains *sticky*, that is the viewer will automatically pan as new vehicle information arrives such that the view center remains with the active vehicle or the vehicle average position. As soon as the user pans manually (with the arrow keys), the viewer breaks from trying to update the view position in relation to received vehicle position information. The rendering of the vehicles can made larger with the '+' key, and smaller with the '-' key, as part of the `VehicleSize` pull-down menu as shown. The size change is applied to all vehicles equally as a scalar multiplier. Currently there is no capability to set the vehicle size individually, or to set the size automatically to scale.

Vehicle trail (track history) rendering can be toggled off and on with the 't' or 'T' key. The default is on. A set of predefined trail colors can be toggled through with the CTRL+'t' key. The individual trail points can be rendered with a line connecting each point, or by just showing the points. When the node report stream is flowing quickly, typically the user doesn't need or want to connect the points. When the viewer is accepting input from an AUV with perhaps a minute or longer delay in between reports, the connecting of points is helpful. This setting can be toggled with the 'y' or 'Y' key, with the default being off. The size of each individual trail point rendering can be made smaller with the '[' key, and larger with the ']' key.

The color of the active vehicle is by default red and can be altered to a handful of other colors in the ActiveColor sub-menu of the Vehicles pull-down menu. Likewise the inactive color, which is by default yellow, can be altered in the InactiveColor sub-menu. These colors can also be altered by setting the `active_vcolor` and `inactive_vcolor` parameters in the `pMarineViewer` configuration block of the MOOS file. They can be set to any color as described in the Colors Appendix.

3.3.4 The "MOOS-Scope" Pull-Down Menu

The "MOOS-Scope" pull-down menu allows the user to configure the `pMarineViewer` to scope on one or more variables in the MOOSDB. The viewer allows visual scoping on only a single variable at a time, but the user can select different variables via the pull-down menu, or toggle between the current and previous variable with the `'/'` key, or cycle between all registered variables with the `CTRL+'/'` key. The scope fields are on the bottom of the viewer as shown in Figures 3 - 6. The three fields show (a) the variable name, (b) the last time is was updated, and (c) the current value of the variable. Configuration of the menu is done in the MOOS configuration block with entries of the following form:

```
SCOPE = <variable>, <variable>, ...
```

The keyword `SCOPE` is not case sensitive, but the MOOS variables are. If no entries are provided in the MOOS configuration block, the pull-down menu contains a single item, the `"Add Variable"` item. By selecting this, the user will be prompted to add a new MOOS variable to the scope list. This variable will then immediately become the actively scoped variable, and is added to the pull-down menu.

3.3.5 The Optional "Action" Pull-Down Menu

The "Action" pull-down menu allows the user to invoke pre-define pokes to the MOOSDB (the MOOSDB to which the pMarineViewer is connected). While hooks for a limited number of pokes are available by configuring on-screen buttons (Section 3.5.2), the number of buttons is limited to four. The "Action" pull-down menu allows for as many entries as will reasonably be shown on the screen. Each action, or poke, is given by a variable-value pair, and an optional grouping key. Configuration is done in the MOOS configuration block with entries of the following form:

```
ACTION = MENU_KEY=<key> # <variable>=<value> # <variable>=<value> # ...
```

If no such entries are provided, this pull-down menu will not appear. The fields to the right of the `ACTION=` are separated by the `'#'` character for convenience to allow several entries on one line. If one wants to use the `'#'` character in one of the variable values, putting double-quotes around the

value will suffice to treat the '#' character as part of the value and not the separator. If the pair has the key word MENU_KEY on the left, the value on the right is a key associated with all variable-value pairs on the line. When a menu selection is chosen that contains a key, then all variable-value pairs with that key are posted to the MOOSDB. If the ACTION key word has a trailing '+' character as below, the pull-down menu will render a line separator after the menu item. The following configuration will result in the pull-down menu depicted in Figure 7.

```
ACTION  = MENU_KEY=deploy # DEPLOY = true # RETURN = false
ACTION+ = MENU_KEY=deploy # MOOS_MANUAL_OVERIDE=false
ACTION  = RETURN=true
```

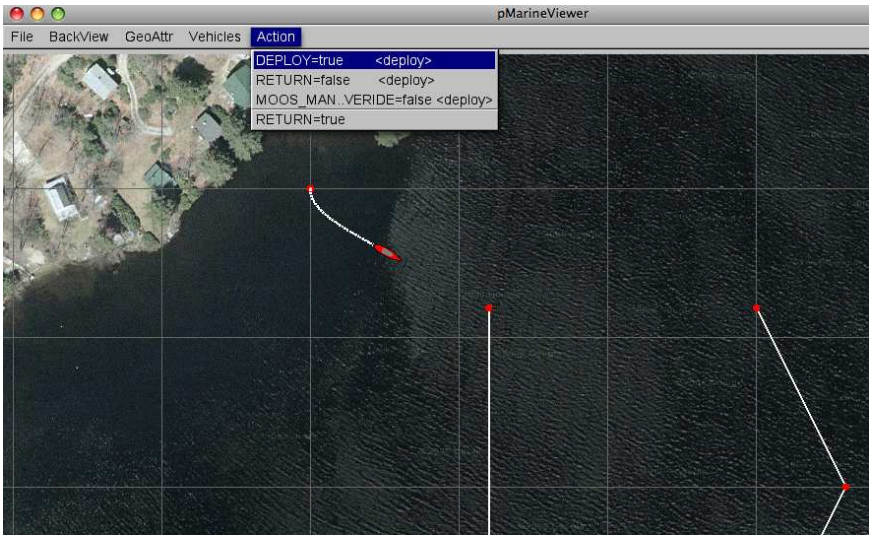


Figure 7: **The Action menu:** The variable value pairs on each menu item may be selected for poking or writing the MOOSDB. The three variable-value pairs above the menu divider will be poked in unison when any of the three are chosen, because they were configured with the same key, <deploy>, shown to the right on each item.

The variable-value pair being poked on an action selection will determine the variable type by the following rule of thumb. If the value is non-numerical, e.g., true, one, it is poked as a string. If it is numerical it is poked as a double value. If one really wants to poke a string of a numerical nature, the addition of quotes around the value will suffice to ensure it will be poked as a string. For example:

```
ACTION   = Vehicle=Nomar # ID="7"
```

As with any other publication to the MOOSDB, if a variable has been previously posted with one type, subsequent posts of a different type will be ignored.

3.3.6 The Optional "Mouse-Context" Pull-Down Menu

When the user clicks the left or right mouse in the geo portion of the pMarineViewer window, the variables MVIEWER_LCLICK and MVIEWER_RCLICK are published respectively with the geo location of the mouse click, and the name of the active vehicle. This is described in more detail in Section 3.5.1. In short a publication of the following is typical:

```
MVIEWER_LCLICK  = "x=82.0,y=-23.0,lat=43.825090305,lon=-70.32937733,vname=Unicorn"
```

The user may further optionally configure pMarineViewer to make additional pokes to the MOOSDB on each left or right mouse click. The user may custom configure the values to allow for the embedding of the operation area position detected under the mouse click. Configuration is done in the MOOS configuration block with entries of the following form:

```
left_context[<key>]  = <var-data-pair>
right_context[<key>] = <var-data-pair>
```

The left_context and right_context keywords are case insensitive. If no such entries are provided, this pull-down menu will not appear. The <key> component is optional and allows for groups of variable-data pairs with the same key to be posted together with the same mouse click. If the <key> is empty, the defined poke will posted on all mouse clicks regardless of the grouping, as is the case with MVIEWER_LCLICK and MVIEWER_RCLICK.

Patterns may be embedded in the string to allow the string to contain information on where the user clicked in the operation area. These patterns are: $(XPOS) and $(YPOS) for the local x and y position respectively, and $(LAT), and $(LON) for the latitude and longitude positions. The pattern $(IX) will expand to an index (beginning with zero by default) that is incremented each time a click/poke is made. This index can be configured to start with any desired index with the lclick_ix_start and rclick_ix_start configuration parameters for the left and right mouse clicks respectively. The following configuration will result in the pull-down menu depicted in Figure 8.

```
left_context[surface_point] = SPOINT = x=$(XPOS), y=$(YPOS), vname=$(VNAME)
left_context[surface_point] = COME_TO_SURFACE = true
left_context[return_point]  = RETURN_POINT = x=$(XPOS), y=$(YPOS), vname=$(VNAME)
left_context[return_point]  = RETURN_HOME = true
left_context[return_point]  = RETURN_HOME_INDEX = $(IX)
right_context[loiter_point] = LOITER_POINT = lat=$(LAT), lon=$(LON)
right_context[loiter_point] = LOITER_MODE = true
```

Note in the figure that the first menu option is "no-action" which shuts off all MOOS pokes associated with any defined groups (keys). In this mode, the MVIEWER_LCLICK and MVIEWER_RCLICK pokes will still be made, along with any other poke configured without a <key>.

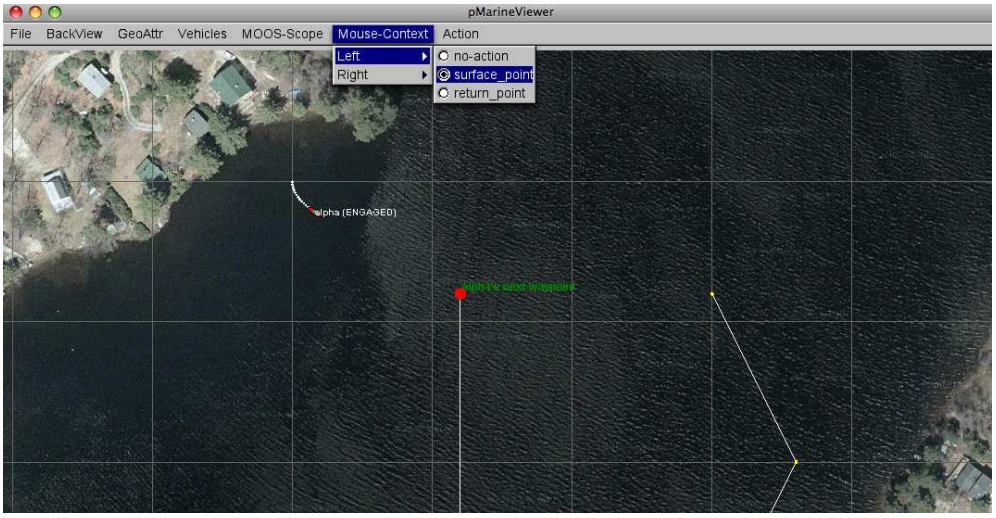


Figure 8: **The Mouse-Context menu:** Keywords selected from this menu will determine which groups of MOOS variables will be poked to the MOOSDB on left or mouse clicks. The variable values may have information embedded indicating the position of the mouse in the operation area at the time of the click.

3.3.7 The Optional "Reference-Point" Pull-Down Menu

The "Reference-Point" pull-down menu allows the user to select a reference point other than the datum, the `(0,0)` point in local coordinates. The reference point will affect the data displayed in the `Range` and `Bearing` fields in the viewer window. This feature was originally designed for field experiments when vehicles are being operated from a ship. An operator on the ship running the `pMarineViewer` would receive position reports from the unmanned vehicles as well as the present position of the ship. In these cases, the ship is the most useful point of reference. Prior versions of this code would allow for a single declaration of the ship name, but the the current version allows for any number of ship names as a possible reference point. This allows the viewer to display the bearing and range between two deployed unmanned vehicles for example. Configuration is done in the MOOS configuration block with entries of the following form:

```
reference_vehicle = vehicle
```

If no such entries are provided, this pull-down menu will not appear. When the menu is present, it looks like that shown in Figure 9. When the reference point is a vehicle with a known heading, the user is able to alter the `Bearing` field from reporting either the relative bearing or absolute bearing. Hot keys are defined for each.

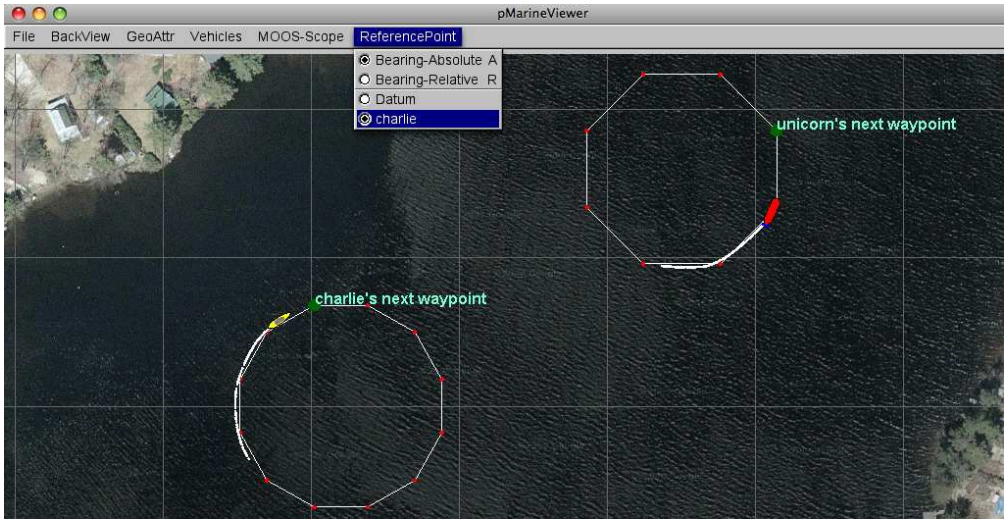


Figure 9: **The Reference-Point menu:** This pull-down menu of the `pMarineViewer` lists the options for selecting a reference point. The reference point determines the values for the `Range` and `Bearing` fields in the viewer for the active vehicle. When the reference point is a vehicle with known heading, the user also may select whether the Bearing is the relative bearing or absolute bearing.

3.4 Displayable Vehicle Shapes, Markers, Drop Points, and other Geometric Objects

The `pMarineViewer` window displays objects in three general categories, (1) the vehicles based on their position reports, (2) markers, which are generally static and things like triangles and squares with labels, and (3) geometric objects such as polygons or lists of line segments that may indicate a vehicle's intended path or other such artifact of it's autonomy situation.

3.4.1 Displayable Vehicle Shapes

The shape rendered for a particular vehicle depends on the *type* of vehicle indicated in the node report received in `pMarineViewer`. There are four types that are currently handled, an AUV shape, a glider shape, a kayak shape, and a ship shape, shown in Figure 10.

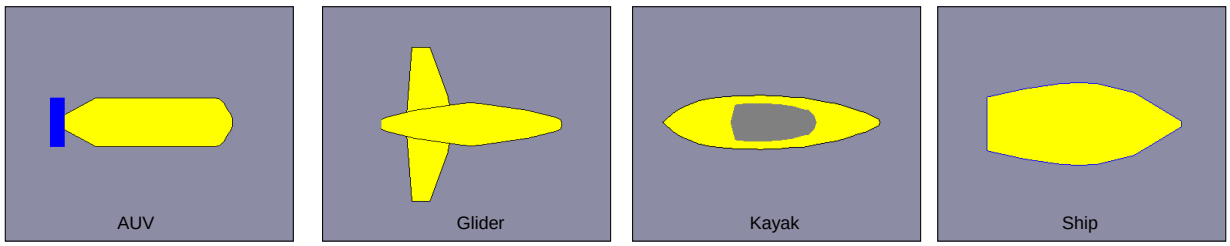


Figure 10: **Vehicles:** Types of vehicle shapes known to the `pMarineViewer`.

The default shape for an unknown vehicle type is currently set to be the shape "ship". The default color for a vehicle is set to be yellow, but can be individually set within the pMarineViewer MOOS configuration block with entries like the following:

```
vehicolor = alpha, turquoise
vehicolor = charlie, navy,
vehicolor = philly, 0.5, 0.9, 1.0
```

The parameter vehicolor is case insensitive, as is the color name. The vehicle name however is case sensitive. All colors of the form described in the Colors Appendix are acceptable.

Mini Exercise #1: Poking a Vehicle into the Viewer.

Issues Explored: (1) Posting a MOOS message resulting in a vehicle rendered in pMarineViewer. (2) Erasing and moving the vehicle.

- Try running the Alpha mission again from the Helm documentation. Note that when the simulation is first launched, a kayak-shaped vehicle sits at position (0,0).
- Use the pMarineViewer MOOS-Scope utility to scope on the variable NODE_REPORT_LOCAL. Either select "Add Variable" from the MOOS-Scope pull-down menu, or type the short-cut key 'a'. Type the NODE_REPORT_LOCAL variable into the pop-up window, and hit Enter. The scope field at the bottom of pMarineViewer should read something like:

  ```
  NODE_REPORT_LOCAL = "NAME=alpha,TYPE=KAYAK,MOOSDB_TIME=2327.07,UTC_TIME=10229133704.48,
  X=0.00,Y=0.00,LAT=43.825300,LON=-70.330400,SPD=0.00,HDG=180.00,YAW=180.00000,DEPTH=0.00,
  LENGTH=4.0,MODE=DISENGAGED,ALLSTOP=ManualOverride
  ```

  This is the posting that resulted in the vehicle currently rendered in the pMarineViewer window. This was likely posted by the pNodeReporter application, but a node report can be poked directly as well to experiment.
- Using the uPokeDB tool, try poking the MOOSDB as follows:

  ```
  $ uPokeDB alpha.moos NODE_REPORT="NAME=bravo,TYPE=glider,X=100,Y=-90,HDG=88,SPD=1.0,
  UTC_TIME=NOW,DEPTH=92,LENGTH=8"
  ```

  Note the appearance of the glider at position (100,-90).

3.4.2 Displayable Marker Shapes

A set of simple static markers can be placed on the geo display for rendering characteristics of an operation area such as buoys, fixed sensors, hazards, or other things meaningful to a user. The six types of markers are shown in Figure 11. They are configured in the pMarineViewer configuration block of the MOOS file with the following format:

```
// Example marker entries in a pMarineViewer config block of a .moos file
// Parameters are case insensitive. Parameter values (except type and color)
// are case sensitive.
marker  = type=efield,x=100,y=20,label=alpha,COLOR=red,width=4.5
marker  = type=square,lat=42.358,lon=-71.0874,color=blue,width=8
```

Each entry is a string of comma-separated pairs. The order is not significant. The only mandatory fields are for the marker type and position. The position can be given in local x-y coordinates

or in earth coordinates. If both are given for some reason, the earth coordinates will take precedent. The `width` parameter is given in meters drawn to scale on the geo display. Shapes are roughly 10x10 meters by default. The GUI provides a hook to scale all markers globally with the 'ALT-M' and 'CTRL-M' hot keys and in the GeoAttributes pull-down menu.

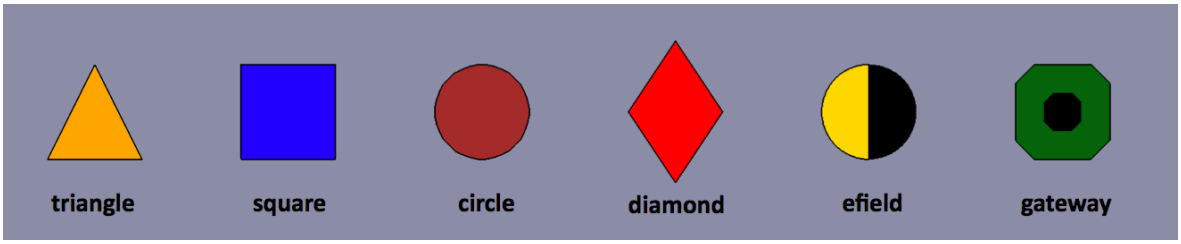


Figure 11: **Markers:** Types of markers known to the `pMarineViewer`.

The color parameter is optional and markers have the default colors shown in Figure 11. Any of the colors described in the Colors Appendix are fair game. The black part of the Gateway and Efield markers is immutable. The label field is optional and is by default the empty string. Note that if two markers of the same type have the same non-empty label, only the first marker will be acknowledged and rendered. Two markers of different types can have the same label.

In addition to declaring markers in the `pMarineViewer` configuration block, markers can be received dynamically by `pMarineViewer` through the `VIEW_MARKER` MOOS variable, and thus can originate from any other process connected to the `MOOSDB`. The syntax is exactly the same, thus the above two markers could be dynamically received as:

```
VIEW_MARKER  = "type=efield,x=100,y=20,SCALE=4.3,label=alpha,COLOR=red,width=4.5"
VIEW_MARKER  = "type=square,lat=42.358,lon=-71.0874,scale=2,color=blue,width=8"
```

The effect of a "moving" marker, or a marker that changes color, can be achieved by repeatedly publishing to the `VIEW_MARKER` variable with only the position or color changing while leaving the label and type the same.

3.4.3 Displayable Drop Points

A user may be interested in determining the coordinates of a point in the geo portion of the `pMarineViewer` window. The mouse may be moved over the window and when holding the `SHIFT` key, the point under the mouse will indicate the coordinates in the local grid. When holding the `CTRL` key, the point under the coordinates are shown in lat/lon coordinates. The coordinates are updated as the mouse moves and disappear thereafter or when the `SHIFT` or `CTRL` keys are release. Drop points may be left on the screen by hitting the left mouse button at any time. The point with coordinates will remain rendered until cleared or toggled off. Each click leaves a new point, as shown in Figure 12.

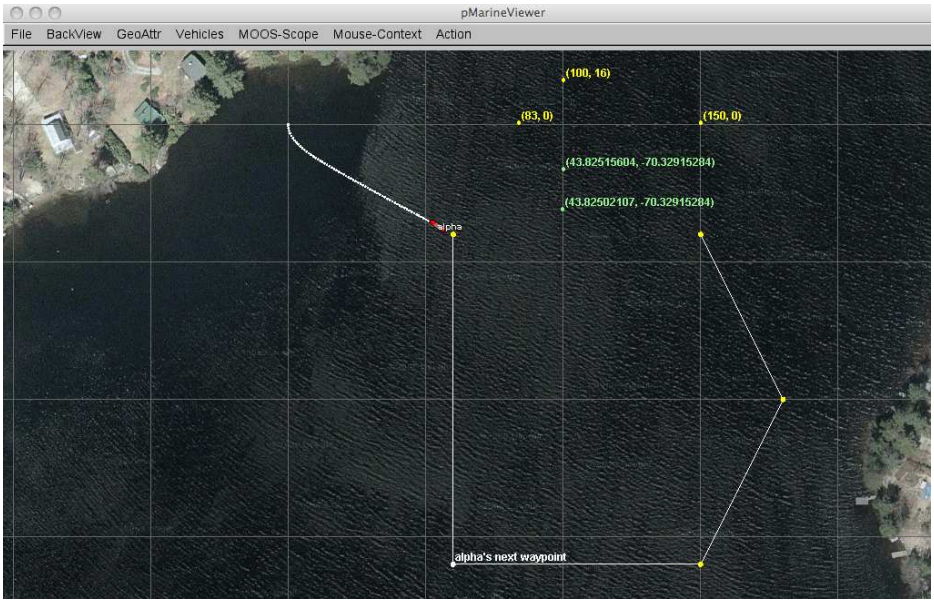


Figure 12: **Drop points:** A user may leave drop points with coordinates on the geo portion of the pMarineViewer window. The points may be rendered in local coordinates or in lat/lon coordinates. The points are added by clicking the left mouse button while holding the `SHIFT` key or `CTRL` key. The rendering of points may be toggled on/off, cleared in their entirety, or reduced by popping the last dropped point.

Parameters regarding drop points are accessible from the `GeoAttr` pull-down menu. The rendering of drop points may be toggled on/off by hitting the 'r' key. The set of drop points may be cleared in its entirety. Or the most recently dropped point may be removed by typing the `CTRL-r` key. The pull-down menu may also be used to change the rendering of coordinates from `"as-dropped"` where some points are in local coordinates and others in lat/lon coordinates, to `"local-grid"` where all coordinates are rendered in the local grid, or `"lat-lon"` where all coordinates are rendered in the lat/lon format.

3.4.4 Displayable Geometric Objects

Some additional objects can be rendered in the viewer such as convex polygons, points, and a set of line segments. In Figures 3 and 4, each vehicle has traversed to and is proceeding around a hexagon pattern. This is apparent from both the rendered hexagon, and confirmed by the trail points. Displaying certain markers in the display can be invaluable in practice to debugging and confirming the autonomy results of vehicles in operation. The intention is to allow for only a few key additional objects to be drawable to avoid letting the viewer become overly specialized and bloated.

In addition to the `NODE_REPORT` variable indicating vehicle pose, `pMarineViewer` registers for the following additional MOOS variables - `VIEW_POLYGON`, `VIEW_SEGLIST`, `VIEW_POINT`. Example values of these variables:

```
VIEW_POLYGON = label,foxtrot:85,-9:100,-35:85,-61:55,-61:40,-35:55,-9
VIEW_POINT   = x=10,y=-80,label=bravo
VIEW_SEGLIST = label,charlie:0,100:50,-35:25,-63
```

Each variable describes a data structure implemented in the geometry library linked to by pMarineViewer. Instances of these objects are initialized directly by the strings shown above. A key member variable of each geometric object is the *label* since pMarineViewer maintains a (C++, STL) map for each object type, keyed on the label. Thus a newly received polygon replaces an existing polygon with the same label. This allows one source to post its own geometric cues without clashing with another source. By posting empty objects, i.e., a polygon or seglist with zero points, or a point with zero radius, the object is effectively erased from the geo display. The typical intended use is to let a behavior within the helm to post its own cues by setting the label to something unique to the behavior. The VIEW_POLYGON listed above for example was produced by a loiter behavior and describes a hexagon with the six points that follow.

3.5 Support for Command-and-Control Usage

For the most part pMarineViewer is intended to be only a receiver of information from the vehicles and the environment. Adding command and control capability, e.g., widgets to re-deploy or manipulate vehicle missions, can be readily done, but make the tool more specialized, bloated and less relevant to a general set of users. A certain degree of command and control can be accomplished by poking key variables and values into the local MOOSDB, and this section describes three methods supported by pMarineViewer for doing just that.

3.5.1 Poking the MOOSDB with Geo Positions

The graphic interface of pMarineViewer provides an opportunity to poke information to the MOOSDB based on visual feedback of the operation area shown in the geo display. To exploit this, two command and control hooks were implemented with a small footprint. When the user clicks on the geo display, the location in local coordinates is noted and written out to one of two variables - MVIEWER_LCLICK for left mouse clicks, and MVIEWER_RCLICK for right mouse clicks, with the following syntax:

```
MVIEWER_LCLICK  = "x=958.0,y=113.0,vname=nyak200",
```

and

```
MVIEWER_RCLICK  = "x=740.0,y=-643.0,vname=nyak200".
```

One can then write another specialized process, e.g., pViewerRelay, that subscribes to these two variables and takes whatever command and control actions desired for the user's needs. One such incarnation of pViewerRelay was written (but not distributed or addressed here) that interpreted the left mouse click to have the vehicle station-keep at the clicked location.

3.5.2 Configuring GUI Buttons for Command and Control

The pMarineViewer GUI can be optionally configured to allow for four push-buttons to be enabled and rendered in the lower-right corner. Each button can be associated with a button label, and a list of variable-value pairs that will be poked to the MOOSDB to which the pMarineViewer process is connected. The basic syntax is as follows:

```
BUTTON_ONE   = <label> # <variable>=VALUE # <variable>=<value>  ...
BUTTON_TWO   = <label> # <variable>=VALUE # <variable>=<value>  ...
BUTTON_THREE = <label> # <variable>=VALUE # <variable>=<value>  ...
BUTTON_FOUR  = <label> # <variable>=VALUE # <variable>=<value>  ...
```

The left-hand side contains one of the four button keywords, e.g., `BUTTON_ONE`. The right-hand side consists of a `'#'`-separated list. Each component in this list is either a `'='`-separated variable-value pair, or otherwise it is interpreted as the button's label. The ordering does not matter and the `'#'`-separated list can be continued over multiple lines as in lines 59-60 in Listing 5 on page 40.

The variable-value pair being poked on a button call will determine the variable type by the following rule of thumb. If the value is non-numerical, e.g., `true`, `one`, it is poked as a string. If it is numerical it is poked as a double value. If one really wants to poke a string of a numerical nature, the addition of quotes around the value will suffice to ensure it will be poked as a string. For example:

```
BUTTON_ONE   = Start # Vehicle=Nomar # ID="7"
```

In this case, clicking the button labeled `"Start"` will result in two pokes, the second of which will have a string value of `"7"`, not a numerical value. As with any poke to the `MOOSDB` of a given variable-value pair, if the value is of a type inconsistent with the first write to the DB under that variable name, it will simply by ignored.

As described in Section 3.3.5, additional variable-value pairs for poking the MOOSDB can be configured in the "Action" pull-down menu. Unlike the use of buttons, which is limited to four, the number of actions in the pull-down menu is limited only by what can reasonably be rendered on the user's screen.

3.6 Configuration Parameters for pMarineViewer

Many of the display settings available in the pull-down menus described in Sections 3.3 can also be set in the `pMarineViewer` block of the MOOS configuration file. Mostly this redundancy is for convenience for a user to have the desired settings without further keystrokes after start-up. An example configuration block is shown in Listing 5.

Parameter	Description	Allowed Values	Default
`hash_view`	Turning off or on the hash marks.	`true, false`	`true`
`hash_delta`	Distance between hash marks	`[10, 1000]`	`50`
`hash_shade`	Shade of hash marks - 0 is black to 1 is white	`[0, 1.0]`	`0.65`
`back_shade`	Shade of hash marks - 0 is black to 1 is white	`[0, 1.0]`	`0.55`
`tiff_view`	Background image used if set to true	`true, false`	`true`
`tiff_type`	Uses the first (A) image if set to `true`	`true, false`	`true`
`tiff_file`	Filename of a tiff file background image	any tiff file	`Default.tif`
`tiff_file_b`	Filename of a tiff file background image	any tiff file	`DefaultB.tif`
`view_center`	The center of the viewing image (the zoom-to point)	`(x, y)`	`(0, 0)`

Table 1: **Background parameters:** Parameters affecting the rendering of the `pMarineViewer` background.

Parameter	Description	Allowed Values	Default
bearing_lines_viewable	Render bearing line objects	true, false, toggle	true
trails_color	Color of points rendered in a trail history	any color	white
trails_connect_viewable	Render lines between dots if true	true, false, toggle	false
trails_history_size	Number of points stored in a trail history	[0, 10,000]	1,000
trails_length	Number of points rendered in a trail history	[0, 10,000]	100
trails_point_size	Size of dots rendered in a trail history	[0, 100]	1
trails_viewable	Trail histories note rendered if false	true, false, toggle	true
vehicles_active_color	Color of the one active vehicle	any color	red
vehicles_inactive_color	Color of other inactive vehicles	any color	yellow
vehicles_name_active	Active vehicle set to the named vehicle	known name	1st
vehicles_name_center	Center vehicle set to the named vehicle	known name	n/a
vehicles_name_color	Color of the font for all vehicle labels	any color	white
vehicles_name_viewable	Vehicle labels not rendered if set to off	off, names, names+mode, names+shortmode, names+depth	names
vehicles_shape_scale	Change size rendering - 1.0 is actual size	[0.1, 100]	1
vehicles_viewable	Vehicles not rendered is set to false	true, false, toggle	false
vehicolor	Override inactive vehicle color individually	See p.x	n/a

Table 2: **Vehicle parameters:** Parameters affecting how vehicles are rendered in pMarineViewer.

Parameter	Description	Allowed Values	Default
marker	Add and newly defined marker	See p. 34	n/a
markers_viewable	If true all markers are rendered	true, false, toggle	true
markers_labels_viewable	If true marker labels are rendered	true, false, toggle	true
markers_scale_global	Marker widths are multiplied by this factor	[0.1, 100]	1
markers_label_color	Color of rendered marker labels	any color	white

Table 3: **Marker parameters:** Parameters affecting the rendering of the pMarineViewer markers.

Parameter	Description	Allowed Values	Default
circle_edge_color	Color rendered circle lines	any color	yellow
circle_edge_width	Line width of rendered circle lines	[0, 10]	2
grid_edge_color	Color of rendered grid lines	any color	white
grid_edge_width	Line width of rendered grid lines	[0, 10]	2
grid_viewable_all	If true grids will be rendered	true, false	true
grid_viewable_labels	If true grid labels will be rendered	true, false	true
point_viewable_all	If true points will be rendered	true, false	true
point_viewable_labels	If true point labels will be rendered	true, false	true
point_vertex_color	Color of rendered points	any color	yellow
point_vertex_size	Size of rendered points	[0, 10]	4
polygon_edge_color	Color of rendered polygon lines	any color	yellow
polygon_edge_width	Line width of rendered polygon edges	[0, 10]	1
polygon_label_color	Color rendered polygon labels	any color	khaki
polygon_viewable_all	If true all polygons are rendered	true, false	true
polygon_viewable_labels	If true polygon labels are rendered	true, false	true
polygon_vertex_color	Color of rendered polygon vertices	any color	red
polygon_vertex_size	Size of rendered polygon vertices	[0, 10]	3
seglist_edge_color	Color or rendered seglist lines	any color	white
seglist_edge_width	Line width of rendered seglist edges	[0,10	1
seglist_label_color	Color of rendered seglist labels	any color	orange
seglist_viewable_all	If true all seglists are rendered	true, false	true
seglist_viewable_labels	If true seglist labels are rendered	true, false	true
seglist_vertex_color	Color of rendered seglist vertices	any color	blue
seglist_vertex_size	Size of rendered seglist vertices	[0, 10]	3

Table 4: **Geometric parameters:** Parameters affecting the rendering of the pMarineViewer geometric objects.

Parameter	Description	Allowed Values	Default
lclick_ix_start	Starting index for left mouse index macro	Any integer	0
rclick_ix_start	Starting index for right mouse index macro	Any integer	0

Table 5: **Other parameters:** Miscellaneous configuration parameters.

Listing 5 - An example pMarineViewer configuration block.

```
1  LatOrigin  =   47.7319
2  LongOrigin = -122.8500
3
4  //---------------------------------------------
5  // pMarineViewer configuration  block
6
7  ProcessConfig = pMarineViewer
8  {
9    // Standard MOOS parameters affecting comms and execution
10   AppTick    = 4
11   CommsTick  = 4
12
13   // Set the background images
```

```
14    TIFF_FILE   = long_beach_sat.tif
14    TIFF_FILE_B = long_beach_map.tif
15
16
17    // Parameters and their default values
18    hash_view     = false
19    hash_delta    = 50
20    hash_shade    = 0.65
21    back_shade    = 0.70
22    trail_view    = true
23    trail_size    = 0.1
24    tiff_view     = true
25    tiff_type     = true
26    zoom          = 1.0
27    verbose       = true
28    vehicles_name_viewable = false
29    bearing_lines_viewable = true
30
31    // Setting the vehicle colors - default is yellow
32    vehicolor     = henry,dark_blue
33    vehicolor     = ike,0.0,0.0,0.545
34    vehicolor     = jane,hex:00,00,8b
35
36    // All polygon parameters are optional - defaults are shown
37    // They can also be set dynamically in the GUI in the GeoAttrs pull-down menu
38    polygon_edge_color   = yellow
39    polygon_vertex_color = red
40    polygon_label_color  = khaki
41    polygon_edge_width   = 1.0
42    polygon_vertex_size  = 3.0
43    polygon_viewable_all = true;
44    polygon_viewable_labels = true;
45
46    // All seglist parameters are optional - defaults are shown
47    // They can also be set dynamically in the GUI in the GeoAttrs pull-down menu
48    seglist_edge_color   = white
49    seglist_vertex_color = dark_blue
50    seglist_label_color  = orange
51    seglist_edge_width   = 1.0
52    seglist_vertex_size  = 3.0
53    seglist_viewable_all = true;
54    seglist_viewable_labels = true;
55
56    // All point parameters are optional - defaults are shown
57    // They can also be set dynamically in the GUI in the GeoAttrs pull-down menu
58    point_vertex_size  = 4.0;
59    point_vertex_color = yellow
60    point_viewable_all = true;
61    point_viewable_labels = true;
62
63    // Define two on-screen buttons with poke values
64    button_one   = DEPLOY # DEPLOY=true
65    button_two   = MOOS_MANUAL_OVERIDE=false # RETURN=false
66    button_two   = RETURN # RETURN=true
67    button_three = DEPTH-10 # OPERATION_DEPTH=10
```

```
68    button_four  = DEPTH-30 # OPERATION_DEPTH=30
69
70    // Declare variable for scoping. Variable names case sensitive
71    scope = PROC_WATCH_SUMMARY
72    scope = BHV_WARNING
73    scope = BHV_ERROR
74
75    // Declare Variable-Value pairs for convenient poking of the MOOSDB
76    action = OPERATION_DEPTH=50
77    action = OPERATION_DEPTH=0 # STATUS="Coming To the Surface"
78  }
```

Color references as in lines 31-33 can be made by name or by hexadecimal or decimal notation. (All three colors in lines 31-33 are the same but just specified differently.) See the Colors Appendix for a list of available color names and their hexadecimal equivalent.

The `VERBOSE` parameter on line 27 controls the output to the console. The console output lists the types of mail received on each iteration of `pMarineViewer`. In the non-verbose mode, a single character is output for each received mail message, with a '*' for `NODE_REPORT`, a 'P' for a `VIEW_POLYGON`, a '.' for a `VIEW_POINT`, and a 'S' for a `VIEW_SEGLIST`. In the verbose mode, each received piece of mail is listed on a separate line and the source of the mail is also indicated. An example of both modes is shown in Listing 6.

Listing 6 - An example pMarineViewer console output.

```
 1    // Example pMarineViewer console output NOT in verbose mode
 2
 3    13.56 > ****..
 4    13.82 > **..
 5    14.08 > **..
 6    14.35 > **..
 7    14.61 > ****.P.P
 8    14.88 > **..
 9    15.14 > **..
10
11    // Example pMarineViewer console output in verbose mode
12
13    15.42 >
14       NODE-REPORT(nyak201)
15       NODE-REPORT(nyak200)
16       Point(nyak201_wpt)
17       Point(nyak200_wpt)
18
19    15.59 >
20       Point(nyak201)
21       Poly(nyak201-LOITER)
22       NODE-REPORT(nyak201)
23       NODE-REPORT(nyak200)
24       Point(nyak200)
25       Poly(nyak200-LOITER)
```

3.7 More about Geo Display Background Images

The geo display portion of the viewer can operate in one of two modes, a grey-scale background, or an image background. Section 3.3.1 addressed how to switch between modes in the GUI interface. To use an image in the geo display, the input to `pMarineViewer` comes in two files, an image file in TIFF format, and an information text file correlating the image to the local coordinate system. The file names should be identical except for the suffix. For example `dabob_bay.tif` and `dabob_bay.info`. Only the `.tif` file is specified in the `pMarineViewer` configuration block of the MOOS file, and the application then looks for the corresponding `.info` file. The info file contains six lines - an example is given in Listing 7.

The geo display portion of the viewer can operate in one of two modes, a grey-scale background, or an image background. Section 3.3.1 addressed how to switch between modes in the GUI interface. To use an image in the geo display, the input to `pMarineViewer` comes in two files. The first is an image file in TIFF format. This file is read by the `lib_tiff` library linked to `pMarineViewer` and, as of this writing, this library requires the image to be square, and the number of pixels in each dimension to be a power of two. The jpeg image patches served from Google Maps for example all this property.

The second file is an information text file correlating the image to the local coordinate system. The file names should be identical except for the suffix. For example `dabob_bay.tif` and `dabob_bay.info`. Only the `.tif` file is specified in the `pMarineViewer` configuration block of the MOOS file, and the application then looks for the corresponding `.info` file. The info file contains six lines - an example is given in Listing 7.

Listing 7 - An example .info file for the pMarineViewer

```
1  // Lines may be in any order, blank lines are ok
2  // Comments begin with double slashes
3
4  datum_lat  = 47.731900
5  datum_lon  = -122.85000
6  lat_north  = 47.768868
7  lat_south  = 47.709761
8  lon_west   = -122.882080
9  lon_east   = -122.794189
```

All four latitude/longitude parameters are mandatory. The two datum lines indicate where $(0, 0)$ in local coordinates is in earth coordinates. However, the datum used by pMarineViewer is determined by the `LatOrigin` and `LongOrigin` parameters set globally in the MOOS configuration file. The datum lines in the above information file are used by applications other than `pMarineViewer` that are not configured from a MOOS configuration file. The `lat_north` parameters correlate the upper edge of the image with its latitude position. Likewise for the other three parameters and boundaries. Two image files may be specified in the `pMarineViewer` configuration block. This allows a map-like image and a satellite-like image to be used interchangeably during use. (Recall the ToggleBackGroundType entry in the BackView pull-down menu discussed earlier.) An example of this is shown in Figure 13 with two images of Dabob Bay in Washington State. Both image files where created from resources at www.maps.google.com.

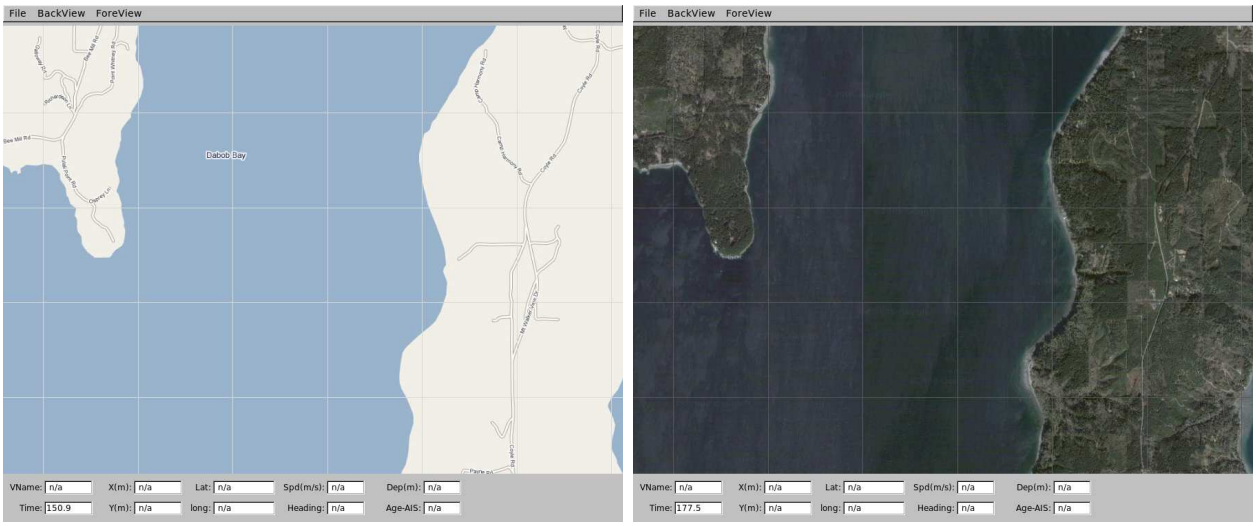


Figure 13: **Dual background geo images:** Two images loaded for use in the geo display mode of `pMarineViewer`. The user can toggle between both as desired during operation.

In the configuration block, the images can be specified by:

```
TIFF_FILE   = dabob_bay_map.tif
TIFF_FILE_B = dabob_bay_sat.tif
```

By default `pMarineViewer` will look for the files `Default.tif` and `DefaultB.tif` in the local directory unless alternatives are provided in the configuration block.

3.8 Publications and Subscriptions for pMarineViewer

3.8.1 Variables published by the pMarineViewer application

Variables published by `pMarineViewer` are summarized in Table 6 below. A more detail description of each variable follows the table.

#	Variable	Description
1	`MVIEWER_LCLICK`	The position in local coordinates of a user left mouse button click
2	`MVIEWER_RCLICK`	The position in local coordinates of a user right mouse button click
3	`HELM_MAP_CLEAR`	A message read by `pHelmIvP` to re-send information on viewer startup.

Table 6: **Variables published by the `pMarineViewer` application.**

Note that `pMarineViewer` may be configured to poke the MOOSDB via either the Action pull-down menu (Section 3.3.5), or via configurable GUI buttons (Section 3.5.2). It may also publish to the MOOSDB variables configured to mouse clicks (Section 3.3.6). So the list of variables that `pMarineViewer` publishes is somewhat user dependent, but the following few variables may be published in all configurations.

- MVIEWER_LCLICK: When the user clicks the left mouse button, the position in local coordinates, along with the name of the active vehicle is reported. This can be used as a command and control hook as described in Section 3.5. As an example:

```
MVIEWER_LCLICK = ‘‘x=-56.0,y=-110.0,vname=alpha’’
```

- MVIEWER_RCLICK: This variable is published when the user clicks with the right mouse button. The same information is published as with the left click.

- HELM_MAP_CLEAR: This variable is published once when the viewer connects to the MOOSDB. It is used in the pHelmIvP application to clear a local buffer used to prevent successive identical publications to its variables.

3.8.2 Variables subscribed for by pMarineViewer application

- BEARING_LINE: A string designation of a bearing from a given vehicle in a given direction. An example:

```
BEARING_LINE = "vname=alpha, bearing=174, range=90, vector_width=1, vector_color=green,
               time_limit=15"
```

or

```
BEARING_LINE = "vname=alpha, bearing=174, range=90, vector_width=1, vector_color=green,
               time_limit=nolimit, bearing_absolute=true"
```

- NODE_REPORT: This is the primary variable consumed by pMarineViewer for collecting vehicle position information. An example:

```
NODE_REPORT = "NAME=nyak201,TYPE=kayak,MOOSDB_TIME=53.049,UTC_TIME=1195844687.236,X=37.49,
              Y=-47.36, SPD=2.40,HDG=11.17,DEPTH=0"
```

- NODE_REPORT_LOCAL: This serves the same purpose as the above variable. In some simulation cases this variable is used.

- TRAIL_RESET: When the viewer receives this variable it will clear the history of trail points associated with each vehicle. This is used when the viewer is run with a simulator and the vehicle position is reset and the trails become discontinuous.

- VIEW_POLYGON: A string representation of a polygon.

- VIEW_POINT: A string representation of a point.

- VIEW_SEGLIST: A string representation of a segment list.

- VIEW_CIRCLE: A string representation of a circle.

- VIEW_MARKER: A string designation of a marker type, size and location.

4 The uXMS Utility: Scoping the MOOSDB from the Console

4.1 Brief Overview

The `uXMS` application is a terminal based tool for live scoping on a MOOSDB process. It is not dependent on any graphic libraries and it is more likely to run out-of-the-box on machines that may not have requisite libraries like FLTK installed. It is easily configured from the command line or a MOOS configuration block to scope on as little as one variable. Listing 8 below shows what the output may look like with the shown command line invocation. The primary use of `uXMS` is to show a snapshot of the MOOSDB for a given subset of variables. A second use is to show the evolving history of a variable which may catch activity that eludes the perspective of a periodic snapshot.

Listing 8 - Example output from uXMS.

```
 1  > uXMS alpha.moos NAV_X NAV_Y NAV_HEADING NAV_SPEED MOOSMANUAL_OVERIDE DEPLOY IVPHELM_ENGAGED
 2
 3    VarName              (S)ource       (T)ime      (C)ommunity   VarValue (MODE = SCOPE:STREAMING)
 4    ----------------     ----------     ----------  ----------    ----------- (89)
 5    NAV_X                uSimMarine     60.47       alpha         59.83539
 6    NAV_Y                uSimMarine     60.47       alpha         -81.67491
 7    NAV_HEADING          uSimMarine     60.47       alpha         -179.77803
 8    MOOS_MANUAL_OVERIDE  pMarineViewer  2.54        alpha         "false"
 9    DEPLOY               pMarineViewer  2.54        alpha         "true"
10    IVPHELM_ENGAGED      pHelmIvP       59.83       alpha         "ENGAGED"
11    NAV_SPEED            uSimMarine     60.47       alpha         2
```

Scoping on the MOOSDB is a very important tool in the process of development and debugging. The `uXMS` tool has a substantial set of configuration choices for making this job easier by bringing just the right data to the user's attention. The default usage, as shown in Listing 8 remains fairly simple, but there are other options discussed in this section that are worth exploiting by the more experienced user.

4.2 The uXMS Refresh Modes

Reports such as the one shown in Listing 8 are generated either automatically or specifically when the user asks for it. The latter is important in situations where bandwidth is low. This feature was the origional motivation for developing `uXMS`. The three *refresh modes* are shown in Figure 14.

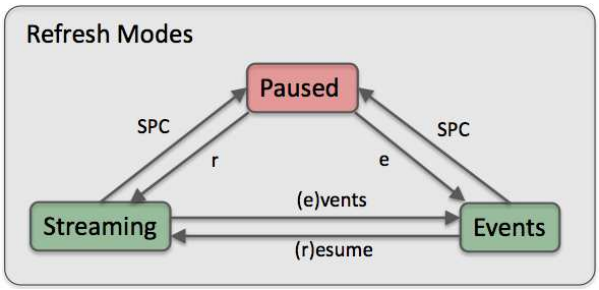


Figure 14: **Refresh Modes:** The `uXMS` refresh mode determines when a new report is written to the screen. The user may switch between modes with the shown keystrokes.

The refresh mode may be changed by the user as `uXMS` is running, or it may be given an initial mode value on startup from the command line with `--mode=paused`, `--mode=events`, or `--mode=streaming`. The latter is the default.

4.2.1 The Streaming Refresh Mode

In the *streaming* refresh mode, the default refresh mode, a new report is generated and written to stdout on every iteration of the `uXMS` application. The frequency is simply controlled by the `AppTick` setting in the MOOS configuration block. The refresh mode is shown in parentheses in the report header as in line 3 of Listing 8. Each report written to the terminal will increment the counter at the end of the line below, e.g., line 4 in Listing 8. The user may re-enter the streaming mode by hitting the `'r'` key.

4.2.2 The Events Refresh Mode

In the *events* refresh mode, a new report is generated only when new mail is received for one of the scoped variables. Note this does not necessarily mean that the value of the variable has changed, only that it has been written to again by some process. This mode is useful in low-bandwidth situations where a user cannot afford the streaming refresh mode, but may be monitoring changes to one or two variables. This mode may be entered by hitting the `'e'` key, or chosen as the initial refresh mode at startup from the command line with the `--mode=events` option.

4.2.3 The Paused Refresh Mode

In the *paused* refresh mode, the report will not be updated until the user specifically requests a new update by hitting the spacebar key. This mode is the preferred mode in low bandwidth situations, and simply as a way of stabilzing the rapid refreshing output of the other modes so one can actually read the output. This mode is entered by the spacebar key and subsequent hits refresh the output once. To launch `uXMS` in the paused mode, use the `--mode=paused` command line switch.

4.3 The uXMS Content Modes

The contents of the `uXMS` report vary between one of a few modes. In the *scoping* mode, a snapshot of a subset of MOOS variables is generated, similar to what is shown in Listing 8. In the *history* mode the recent history of changes to a single MOOS variable is reported.

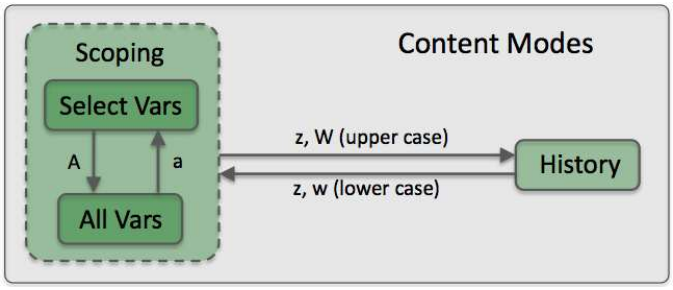


Figure 15: **Content Modes:** The `uXMS` content mode determines what data is included in each new report. The two major modes are the *scoping* and *history* modes. In the former, snapshots of one or more MOOS variables are reported. In the latter, the recent history of a single variable is reported.

4.3.1 The Scoping Content Mode

The *scoping* mode has two sub-modes as shown in Figure 15. In the first sub-mode, the *SelectVars* sub-mode, the only variables shown are the ones the user requested. They are requested on the command-line upon start-up (Section 4.5), or in the `uXMS` configuration block in the `.moos` file provided on startup, or both. One may also select variables for viewing by specifying one or MOOS processes with the command line option `--src=<process>,<process>,...`. All variables from these processes will then be included in the scope list.

In the *AllVars* sub-mode, all MOOS variables in the MOOSDB are displayed, unless explicitly filtered out. The most common way of filtering out variables in the *AllVars* sub-mode is to provide a filter string interactively by typing the `'/'` key and entering a filter. Only lines that contain this string as a substring in the variable name will then be shown. The filter may also be provided on startup with the `--filter=pattern` command line option.

In both sub-modes, variables that would otherwise be included in the report may be masked out with two further options. Variables that have never been written to by any MOOS process are referred to as virgin variables, and by default are shown with the string `"n/a"` in their value column. These may be shut off from the command line with `--mask=virgin`, or in the MOOS configuration block by including the line `DISPLAY_VIRGINS=false`. Similarly, variables with an empty string value may be masked out from the command line with `--mask=empty`, or with the line `DISPLAY_EMPTY_STRINGS=false` in the MOOS configuration block of the `.moos` file.

4.3.2 The History Content Mode

In the *history* mode, the recent values for a single MOOS variable are reported. Contrast this with the *scoping* mode where a snapshot of a variable value is displayed, and that value may have changed several times between successive reports. The output generated in this mode may look like the following, in Listing 9, which captures a vehicle going into a turn after a long period of near steady heading:

Listing 9 - Example report output in the history content mode of uXMS.

```
 0   VarName              (S)ource    (T)ime       VarValue (MODE = HISTORY:PAUSED)
 1   -----------------    ----------  ----------   ----------- (132)
 2   DESIRED_HEADING      pHelmIvP    1721.70     (43) 115
 3   DESIRED_HEADING      pHelmIvP    1728.72     (28) 114
 4   DESIRED_HEADING      pHelmIvP    1733.99     (21) 115
 5   DESIRED_HEADING      pHelmIvP    1738.75     (19) 114
 6   DESIRED_HEADING      pHelmIvP    1740.00     (5) 115
 7   DESIRED_HEADING      pHelmIvP    1740.25     (1) 148
 8   DESIRED_HEADING      pHelmIvP    1740.50     (1) 149
 9   DESIRED_HEADING      pHelmIvP    1740.75     (1) 150
10   DESIRED_HEADING      pHelmIvP    1741.00     (1) 152
11   DESIRED_HEADING      pHelmIvP    1741.25     (1) 153
12   DESIRED_HEADING      pHelmIvP    1741.50     (1) 154
13   DESIRED_HEADING      pHelmIvP    1741.75     (1) 156
14   DESIRED_HEADING      pHelmIvP    1742.00     (1) 157
15   DESIRED_HEADING      pHelmIvP    1742.25     (1) 158
16   DESIRED_HEADING      pHelmIvP    1742.50     (1) 160
17   DESIRED_HEADING      pHelmIvP    1742.75     (1) 161
18   DESIRED_HEADING      pHelmIvP    1743.01     (1) 163
```

```
19   DESIRED_HEADING       pHelmIvP    1743.26     (1) 164
20   DESIRED_HEADING       pHelmIvP    1743.51     (1) 167
21   DESIRED_HEADING       pHelmIvP    1743.76     (1) 169
```

The output structure in the *history* mode is the same as in the *scoping* mode in terms of what data is in the columns and header lines. Each line however is dedicated to the same variable and shows the progression of values through time. To save screen real estate, successive mail received for with identical source and value will consolidated on one line, and the number in parentheses is merely incremented for each such identical mail. For example, on line 5 in Listing 9, the value of `DESIRED_HEADING` has remained the same for 19 consecutives posts to the MOOSDB.

The output in the *history* mode may be adjusted in a few ways. The number of lines of history may be increased or descreased by hitting the `'>'` or `'<'` keys respectively. A maximum of 100 and minimum of 5 lines is allowed. To increase the available real estate on each line, the variable name column my be supressed by hitting the `'j'` key and restored with the `'J'` key.

4.4 Configuration File Parameters for uXMS

Configuraton of `uXMS` may be done from a configuration file (`.moos` file), from the command line, or both. Generally the parameter settings given on the command line override the settings from the `.moos` file, but using the configuration file is a convenient way of ensuring certain settings are in effect on repeated command line invocations. The following is short description of the parameters:

Listing 4.10: Configuration Parameters for uXMS.

`COLORMAP:`	Associates a color for the line of text reporting the given variable.
`CONTENT_MODE:`	Determines if the content mode is `scoping` or `history`.
`DISPLAY_ALL:`	If `true`, all variables are reported in the *scoping* content mode.
`DISPLAY_COMMUNITY:`	If `true`, the Community column is rendered.
`DISPLAY_EMPTY_STRINGS:`	if `false`, variables with an empty-string value are not reported.
`DISPLAY_SOURCE:`	If `true`, the Source column is rendered.
`DISPLAY_TIME:`	If `true`, the Time column is rendered.
`DISPLAY_VIRGINS:`	If `false`, variables never written to the MOOSDB are not reported.
`HISTORY_VAR:`	Names the MOOS variable reported in the *history* mode.
`REFRESH_MODE:`	Determines when new reports are written to the screen.
`VAR:`	A comma-separated list of variables to scope on in the *scoping* mode.

A design goal of `uXMS` is to allow the user to customize just the right window into the MOOSDB to facilitate debugging and analysis. Most of the the configurable options deal with content and layout of the information in the terminal window, but color can also be used to faciliate monitoring one or more variables. The parameter

```
COLORMAP = <variable/app>, <color>
```

is used to request that a line the report containing the given variable or produced by the given MOOS application (source) is rendered in the given color. The choices for color are limited to red, green, blue, cyan, and magenta.

The content mode determines what information is generated in each report to the terminal output (Section 4.3). This mode is set with the following parameter:

```
CONTENT_MODE = <mode-type>   // mode-type is set to either "scoping" or "history"
```

The default content mode is the *scoping* mode, but may alternately be set to the *history* mode. These were described in Sections 4.3.1 and 4.3.2.

The `uXMS` report has three columns of data that may be optionally turned off to conserve real estate, the Time, Source and Community columns as shown in Listing 8. By default they are turned off, and they may be toggled on and off by the user at run time. Their initial state may be configured with the following three parameters:

```
DISPLAY_COMMUNITY = <Boolean>
DISPLAY_SOURCE = <Boolean>
DISPLAY_TIME = <Boolean>
```

The report content may be further modified to mask out lines containing variables that have never been written to, and variables with an empty-string value. This is done with the following configuration lines:

```
DISPLAY_VIRGINS = <Boolean>
DISPLAY_EMPTY_STRINGS = <Boolean>
```

The variable reported in the *history* mode is set with the below configuration line:

```
HISTORY_VAR = <MOOS-variable>
```

The history report only allows for one variable, and multiple instances of the above line will simply honor the last line provided. The variables reported on in the *scoping* mode, the scope list, are declared with configuration lines of the form:

```
VAR = <MOOS-variable>, <MOOS-variable>, ...
```

Multiple such lines, each perhaps with multiple variables, are accommodated. The scope list may be *augmented* on the command line by simply naming variables as command line arguments. The scope list provided on the command line may *replace* the list given in the configuration file if the `--clean` command line option is also invoked.

The *refresh* mode determines when new reports are generated to the screen, as discussed in Section 4.2. It is set with the below configuration line:

```
REFRESH_MODE = <mode>  // Valid modes are "paused", "streaming", "events"
```

The initial refresh mode is set to "events" by default. The refresh mode set in the configuration file may be overridden from the command line (Section 4.5), and toggled by the user at run time (Section 4.6).

An example configuration is given in Listing 11.

Listing 11 - An example uXMS configuration block.

```
 1  //------------------------------------------------------------------
 2  // uXMS configuration block
 3
 4  ProcessConfig = uXMS
 5  {
 6    AppTick    = 5
 7    CommsTick  = 5
 8
 9    // Navigation information (Or use --group=nav on the command line)
10    VAR  = NAV_X, NAV_Y, NAV_HEADING, NAV_SPEED, NAV_DEPTH
11
12    // Helm output information (Or use --group=helm on the command line)
13    VAR  = DESIRED_HEADING, DESIRED_SPEED, DESIRED_DEPTH
14
15    // More helm information (Or use --group=helm on the command line)
16    VAR  = BHV_WARNING, BHV_ERROR, MOOS_MANUAL_OVERIDE
17    VAR  = HELM_IPF_COUNT, HELM_ACTIVE_BHV, HELM_NONIDLE_BHV
18    VAR  = DEPLOY, RETURN, STATION_KEEP
19
20    // PID output information (or use --group=pid on the command line)
21    VAR  = DESIRED_RUDDER, DESIRED_THRUST, DESIRED_ELEVATOR
22
23    // uProcessWatch output (or use --group=proc on the command line)
24    VAR  = PROC_WATCH_SUMMARY, PROC_WATCH_EVENT
25
26    // Display parameters - Values shown are defaults
26    REFRESH_MODE          = events
27    DISPLAY_VIRGINS       = true    // false if --mask=virgin on cmd-line
28    DISPLAY_EMPTY_STRINGS = true    // false if --mask=empty  on cmd-line
29    DISPLAY_SOURCE        = false   // true  if --show=source on cmd-line
30    DISPLAY_TIME          = false   // true  if --show=time   on cmd-line
31    DISPLAY_COMMUNITY     = false   // true  if --show=community cmd-line
32
33    // Enable History Scoping by specifying a history variable
34    HISTORY_VAR  = VIEW_POINT
35  }
```

4.5 Command Line Usage of uXMS

Many of the parameters available for setting the `.moos` file configuration block can also be affected from the command line. The command line configurations always trump any configurations in the `.moos` file. As with the `uPokeDB` application, the server host and server port information can be specified from the command line too to make it easy to pop open a `uXMS` window from anywhere within the directory tree without needing to know where the `.moos` file resides. The basic command

line usage for the uXMS application is the following:

Listing 12 - Command line usage for the uXMS *tool.*

```
 0  > uXMS -h
 1  Usage: uXMS [file.moos] [OPTIONS]
 2
 3  Options:
 4    --all,-a    Show ALL MOOS variables in the MOOSDB
 5    --clean,-c  Ignore scope variables in file.moos
 6    --colormap=VAR,color
 7                Display all entries where the variable, source,
 8                or community contains VAR as substring. Color
 9                may be blue, red, magenta, cyan, or green.
10    --colorany=VAR,VAR,...,VAR
11                Display all entries where the variable, source,
12                or community contains VAR as substring. Color
13                chosen automatically from unused colors.
14    --group=[helm,pid,proc,nav]
15                helm: Auto-subscribe for IvPHelm variables
16                pid:  Auto-subscribe for PID (DESIRED_*) vars
17                proc: Auto-subscribe for uProcessWatch vars
18                nav:  Auto-subscribe for NAV_* variables
19    --history=variable
20                Allow history-scoping on variable
21    --mask=[virgin,empty]
22                virgin: Don't display virgin variables
23                empty:  Don't display empty strings
24    --mode=[paused,events,STREAMING]
25                Determine display mode. Paused: scope updated
26                only on user request. Events: data updated only
27                on change to a scoped variable. Streaming: data
28                updated continuously on each app-tick.
29    --server_host=value
30                Connect to MOOSDB at IP=value rather than
31                getting the info from file.moos.
32    --server_port=value
33                Connect to MOOSDB at port=value rather than
34                getting the info from file.moos.
35    --show=[source,time,community,aux]
36                Turn on data display in the named column,
37                source, time, or community. All off by default
38                Enabling aux shows the auxilliary source in
39                the source column.
40    --src=pSrc,pSrc,...,pSrc
41                Scope only on vars posted by the given list of
42                MOOS processes, i.e., sources
43    --trunc=value [10,1000]
44                Truncate the output in the data column.
45
```

The -nav, -pid, -helm, and -proc switches are convenient aliases for common groups of variables. See Listing 11. Using the --clean switch will cause uXMS to ignore the variables specified in the .moos file configuration block and only scope on the variables specified on the command line (otherwise the union of the two sets of variables is used). Typically this is done when a user wants to quickly

scope on a couple variables and doesn't want to be distracted with the longer list specified in the .moos file. Arguments on the command line other than the ones described above are treated as variable requests. Thus the following command line entry:

```
>  uXMS foo.moos --group=proc --clean DB_CLIENTS
```

would result in a scope list of PROC_WATCH_SUMMARY, PROC_WATCH_EVENT and DB_CLIENTS, regardless of what may have been specified in the uXMS configuration block of foo.moos.

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

```
> uXMS DB_CLIENTS --server_host=18.38.2.158  --server_port=9000
```

If the server_host or the server_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 is when the MOOSDB is running on the local machine ("localhost") with port 9000, these are the default values and the user can simply hit the return key.

```
> uXMS DEPLOY DB_CLIENTS    // The latter two args are MOOS variables to scope
> Enter Server: [localhost]
>    The server is set to "localhost"
> Enter Port: [9000] 9123
>    The port is set to "9123"
```

4.6 Console Interaction with uXMS at Run Time

When uXMS is launched, a separate thread is spawned to accept user input at the console window. When first launched the entire scope list is printed to the console in a five column report. The first column displays the variable name, and the last one displays the variable value as shown in Listing 13. Each time the report is written the counter at the end of line 2 is incremented. The variable *type* is indicated by the presence or lack of quotes around the value output. Quotes indicate a string type as in line 3, and lack of quotes indicate a double. A variable value of n/a indicates the variable has yet to be published to the MOOSDB by any process as in lines 8 and 11. Should a variable actually have the value of n/a as a string, it would have quotes around it.

Listing 13 - The uXMS console output with -proc, -nav and -pid command line options.

```
 1    VarName               (S)  (T)  (C) VarValue (MODE = SCOPING:PAUSED)
 2    ----------------      ---  ---  ---  ----------- (1)
 3    PROC_WATCH_SUMMARY                  "All Present"
 4    NAV_X                               10
 5    NAV_Y                               -10
 6    NAV_HEADING                         180
 7    NAV_SPEED                           0
 8    NAV_DEPTH                           n/a
 9    DESIRED_RUDDER                      0
10    DESIRED_THRUST                      0
11    DESIRED_ELEVATOR                    n/a
```

By default at start-up, uXMS is in a paused mode and the three middle columns are un-expanded. Listing 14 shows the console help menu which can be displayed at any time by typing 'h'. Displaying the help menu automatically puts the program into a paused mode if it wasn't already. A common usage pattern to minimize bandwidth is to remain in paused mode and hit the space bar or 'u/U' key to get a single updated report. This action also results in the replacement of the help menu if it is currently displayed, with a new report. A streaming mode is entered by hitting the 'r/R' key, and a report is generated once every iteration of uXMS, the frequency being determined by the MOOS AppTick setting on line 6 in Listing 11. Variables the have never been written to in the MOOSDB ("virgin variables") have a VarValue field of "n/a". Virgin variables can be suppressed by hitting the 'v' key, and by default are displayed. String variable that have an empty string value can also be suppressed by hitting the 'e' key and are also displayed by default.

Listing 14 - The help-menu on the uXMS console.

```
 1  KeyStroke    Function
 2  ---------    --------------------------
 3      s        Suppress Source of variables
 4      S        Display Source of variables
 5      t        Suppress Time of variables
 6      T        Display Time of variables
 7      c        Suppress Community of variables
 8      C        Display Community of variables
 9      v        Suppress virgin variables
10      V        Display virgin variables
11      n        Suppress null/empty strings
12      N        Display null/empty strings
13      w        Show Variable History if enabled
14      W        Hide Variable History
15      z        Toggle the Variable History mode
16      j        Truncate Hist Variable in Hist Report
17      J        Show Hist Variable in Hist Report
18    > or <     Show More or Less Variable History
19      /        Begin entering a filter string
20      ?        Clear current filter
21      a        Revert to variables shown at startup
22      A        Display all variables in the database
23   u/U/SPC     Update infor once-now, then Pause
24     p/P       Pause information refresh
25     r/R       Resume information refresh
26     e/E       Information refresh is event-driven
27     h/H       Show this Help msg - 'R' to resume
```

The three middle columns can be expanded as shown in Listing 15. Column 2 can be activated by typing 'S' and deactivated by 's' and shows the variable source, i.e., the latest process connected to the MOOSDB to post a value to that variable. The third column shows the time (since MOOSDB start-up) of the last write to that variable. uXMS subscribes to the variable DB_UPTIME and reads this mail first and assigns this time stamp to all other incoming mail in that iteration. Time display is activated with 'T' and deactivated with 't'. The fourth column shows the MOOS community of the last variable posting. Unless an inter-MOOSDB communications process is running such as pMOOSBridge or MOOSBlink, entries in this column will be the local community, set by the parameter of the same name in global section of the MOOS file. Output in this column is activated

with 'C' and deactivated with 'c'.

Listing 15 - An example uXMS console output with all fields expanded.

```
1    VarName              (S)ource      (T)ime      (C)ommunity   VarValue (MODE = SCOPING:PAUSED)
2    ----------------     ----------    ----------  ----------     ----------- (4)
3    PROC_WATCH_SUMMARY   uProcessWatch 364.486     nyak200       "AWOL: pEchoVar"
4    NAV_X                pEchoVar      365.512     nyak200       10
5    NAV_Y                pEchoVar      365.512     nyak200       -10
6    NAV_HEADING          pEchoVar      365.512     nyak200       180
7    NAV_SPEED            pEchoVar      365.512     nyak200       0
8    DESIRED_RUDDER       pMarinePID    365.512     nyak200       0
9    DESIRED_THRUST       pMarinePID    365.512     nyak200       0
```

Variables that have yet to be written, as lines 8 and 11 in Listing 13, can be suppressed by the hitting 'v' key, and restored by the 'V' key. In the paused mode, each change in report format has the side-effect of requesting a new report reflecting the desired change in format. The decision was made to use the upper and lower case keys for toggling format features rather simply using the the 's' key for toggling off and on, which was the case on the first version of `uXMS`. In high latency, low bandwidth use, toggling with one key can be leave the user wondering which state is active.

4.7 Running uXMS Locally or Remotely

The choice of `uXMS` as a scoping tool was designed in part to support situations where the target MOOSDB is running on a vehicle with low bandwidth communications, such as an AUV sitting on the surface with only a weak RF link back to the ship. There are two distinct ways one can run `uXMS` in this situation and its worth noting the difference. One way is to run `uXMS` locally on one's own machine, and connect remotely to the MOOSDB on the vehicle. The other way is to log onto the vehicle through a terminal, run `uXMS` remotely, but in effect connecting locally to the MOOSDB also running on the vehicle.

The difference is seen when considering that `uXMS` is running three separate threads. One accepts mail delivered by the MOOSDB, one executes the iterate loop of `uXMS` where reports are written to the terminal, and one monitors the keyboard for user input. If running `uXMS` locally, connected remotely, even though the user may be in paused mode with no keyboard interaction or reports written to the terminal, the first thread still may have a communication requirement perhaps larger than the bandwidth will support. If running remotely, connected locally, the first thread is easily supported since the mail is communicated locally. Bandwidth is consumed in the second two threads, but the user controls this by being in paused mode and requesting new reports judiciously.

4.8 Connecting multiple uXMS processes to a single MOOSDB

Multiple versions of `uXMS` may be connected to a single MOOSDB. This is to simultaneously allow several people a scope onto a vehicle. Although MOOS disallows two processes of the same name to connect to MOOSDB, `uXMS` generates a random number between 0-999 and adds it as a suffix to the `uXMS` name when connected. Thus it may show up as something like `uXMS_871` if you scope on the variable `DB_CLIENTS`. In the unlikely event of a name collision, the user can just try again.

4.9 Publications and Subscriptions for uXMS

Variables published by the uXMS application

- `None`

Variables subscribed for by the uXMS application

- `USER-DEFINED`: The variables subscribed for are those on the *scope list* described in Section 4.4.

5 uTermCommand: Poking the MOOSDB with Pre-Set Values

5.1 Brief Overview

The uTermCommand application is a terminal based tool for poking the MOOS database with pre-defined variable-value pairs. This can be used for command and control for example by setting variables in the MOOSDB that affect the behavior conditions running in the helm. One other way to do this, perhaps known to users of the iRemote process distributed with MOOS, is to use the Custom Keys feature by binding variable-value pairs to the numeric keys [0-9]. The primary drawback is the limitation to ten mappings, but the uTermCommand process also allows more meaningful easy-to-remember cues than the numeric keys.

5.2 Configuration Parameters for uTermCommand

The variable-value mappings are set in the uTermCommand configuration block of the MOOS file. Each mapping requires one line of the form:

```
CMD = cue --> variable --> value
```

The *cue* and *variable* fields are case sensitive, and the *value* field may also be case sensitive depending on how the subscribing MOOS process(es) handle the value. An example configuration is given in Listing 16.

Listing 16 - An example uTermCommand configuration block.

```
 1  //---------------------------------------------------------
 2  // uTermCommand configuration block
 4
 5  ProcessConfig = uTermCommand
 6  {
 7    AppTick    = 2
 8    CommsTick  = 2
 9
10    CMD  = deploy_true   -->  DEPLOY       --> true
11    CMD  = deploy_false  -->  DEPLOY       --> false
12    CMD  = return_true   -->  RETURN       --> true
13    CMD  = return_false  -->  RETURN       --> false
14    CMD  = station_true  -->  STATION_KEEP --> true
15    CMD  = station_false -->  STATION_KEEP --> false
16  }
```

Recall the *type* of a MOOS variable is either a string or double. If a variable has yet to be posted to the MOOSDB, it accepts whatever type is first written, otherwise postings of the wrong type are ignored. In the uTermCommand configuration lines such as 10-15 in Listing 16, the variable type is interpreted to be a string if quotes surround the entry in the value field. If not, the value is inspected as to whether it represents a numerical value. If so, it is posted as a double. Otherwise it is posted as a string. Thus **true** and **"true"** are the same type (no such thing as a Boolean type), **25** is a double and **"25"** is a string.

5.3 Console Interaction with uTermCommand at Run Time

When uTermCommand is launched, a separate thread is spawned to accept user input at the console window. When first launched the entire list of cues and the associated variable-value pairs are listed. Listing 17 shows what the console output would look like given the configuration parameters of Listing 16. Note that even though quotes were not necessary in the configuration file to clarify that **true** was to be posted as a string, the quotes are always placed around string values in the terminal output.

Listing 17 - Console output at start-up.

```
 1    Cue                  VarName                VarValue
 2    -----------          ---------------        --------------
 3    deploy_true          DEPLOY                 "true"
 4    deploy_false         DEPLOY                 "false"
 5    return_true          RETURN                 "true"
 6    return_false         RETURN                 "false"
 7    station_true         STATION_KEEP           "true"
 8    station_false        STATION_KEEP           "false"
 9
10  >
```

A prompt is shown on the last line where user key strokes will be displayed. As the user types characters, the list of choices is narrowed based on matches to the cue. After typing a single 'r' character, only the `return_true` and `return_false` cues match and the list of choices shown are reduced as shown in Listing 18. At this point, hitting the TAB key will complete the input field out to `return_`, much like tab-completion works at a Linux shell prompt.

Listing 18 - Console output after typing a single character 'r'.

```
 1    Cue                  VarName                VarValue
 2    -----------          ---------------        --------------
 3    return_true          RETURN                 "true"
 4    return_false         RETURN                 "false"
 5
 6  > r
```

When the user has typed out a valid cue that matches a single entry, only the one line is displayed, with the tag `<-- SELECT` at the end of the line, as shown in Listing 19.

Listing 19 - Console output when a single command is identified.

```
 1    Cue                  VarName                VarValue
 2    -----------          ---------------        --------------
 3    return_true          RETURN                 "true" <-- SELECT
 4
 5  > return_true
```

At this point hitting the ENTER key will execute the posting of that variable-value pair to the MOOSDB, and the console output will return to its original start-up output. A local history is augmented after each entry is made, and the up- and down-arrow keys can be used to select and re-execute postings on subsequent iterations.

5.4 More on uTermCommand for In-Field Command and Control

The uTermCommand utility can be used in conjunction with a inter-MOOSDB communications utility such as pMOOSBridge or MOOSBlink to effectively control a set of vehicles in the field running the IvP Helm. The user uses uTermCommand to alter a key variable in the local MOOSDB, and this variable gets mapped to one or more vehicles at different IP addresses in the network, sometimes changing variables names in the mapping. The helm is running on the vehicles with one or more behaviors composed with a condition affected by the newly changed variable in its local MOOSDB. The idea is depicted Figure 16.

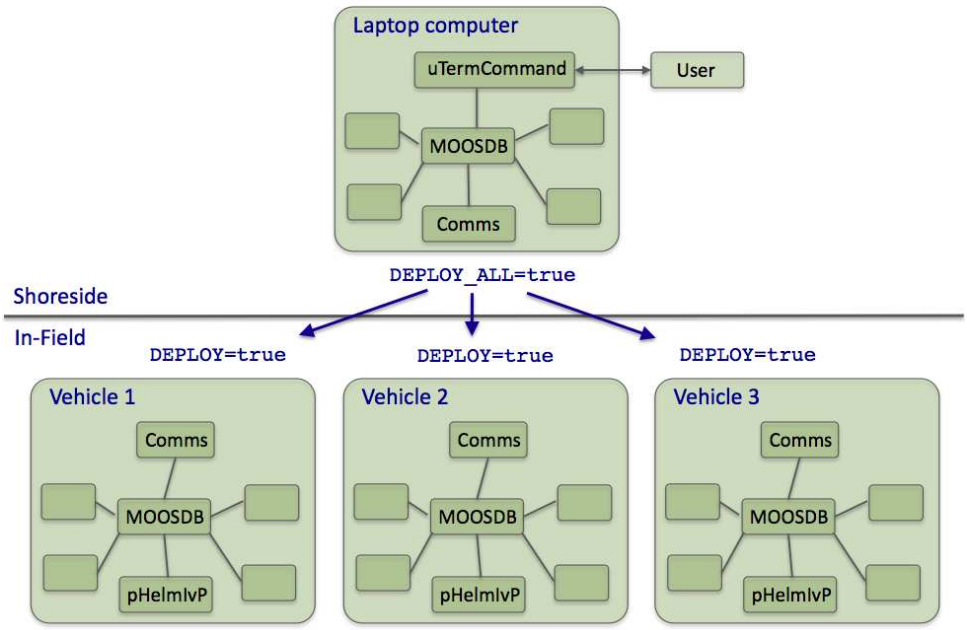


Figure 16: A common usage of the uTermCommand application for command and control.

In the example below in Listing 20, uTermCommand is used to control a pair of vehicles in one of two ways - to deploy a vehicle on a mission, or to recall it to a return point. The configuration block contains three groups. The first group, lines 10-13, are for affecting commands to both vehicles at once, and the second two groups in lines 15-18 and lines 20-23 are for affecting commands to a particular vehicle.

Listing 20 - An example uTermCommand configuration block.

```
1  //------------------------------------------------------------
2  // uTermCommand configuration block
4
5  ProcessConfig = uTermCommand
6  {
7    AppTick    = 2
8    CommsTick  = 2
9
10   CMD  = all_deploy_true    -->  DEPLOY_ALL        --> true
11   CMD  = all_deploy_false   -->  DEPLOY_ALL        --> false
12   CMD  = all_return_true    -->  RETURN_ALL        --> true
```

```
13    CMD  = all_return_false   -->  RETURN_ALL        --> false
14
15    CMD  = 200_deploy_true    -->  DEPLOY_200        --> true
16    CMD  = 200_deploy_false   -->  DEPLOY_200        --> false
17    CMD  = 200_return_true    -->  RETURN_200        --> true
18    CMD  = 200_return_false   -->  RETURN_200        --> false
19
20    CMD  = 201_deploy_true    -->  DEPLOY_201        --> true
21    CMD  = 201_deploy_false   -->  DEPLOY_201        --> false
22    CMD  = 201_return_true    -->  RETURN_201        --> true
23    CMD  = 201_return_false   -->  RETURN_201        --> false
24   }
```

The variable-value postings made by uTermCommand are made in the local MOOSDB and need to be communicated out to the vehicles to have a command and control effect. A few tool exist for this depending on the communications environment (wifi-802.11, radio-frequency, acoustic underwater communications, or local network in simulation mode, etc.). The configuration blocks for two tools, pMOOSBridge and MOOSBlink are shown in Listing 21. Note that each has a way of communicating with several vehicles at once with one variable change - lines 8-9 in MOOSBlink and lines 22-25 in pMOOSBridge. In this way the uTermCommand user can deploy or recall all vehicles with one command. Communication with a single vehicle is set up with lines 11-14 and lines 27-30.

Listing 21 - An example MOOSBlink and pMOOSBridge configuration block for implementing simple command and control with two vehicles.

```
 1  //------------------------------------------------------------
 2  // pMOOSBlink config block
 3
 4  ProcessConfig = MOOSBlink
 5  {
 6    BroadcastAddr = 192.168.1.255
 7
 8    Share = global,  DEPLOY_ALL, DEPLOY, 1
 9    Share = global,  RETURN_ALL, RETURN, 1
10
11    Share = nyak200, DEPLOY_200, DEPLOY, 1
12    Share = nyak201, DEPLOY_201, DEPLOY, 1
13    Share = nyak200, RETURN_200, RETURN, 1
14    Share = nyak201, RETURN_201, RETURN, 1
15  }
16
17  //------------------------------------------------------------
18  // pMOOSBridge config block
19
20  ProcessConfig = pMOOSBridge
21  {
22    SHARE = [DEPLOY_ALL]  -> nyak200 @ 192.168.0.200:9000 [DEPLOY]
23    SHARE = [DEPLOY_ALL]  -> nyak201 @ 192.168.0.201:9000 [DEPLOY]
24    SHARE = [RETURN_ALL]  -> nyak200 @ 192.168.0.200:9000 [RETURN]
25    SHARE = [RETURN_ALL]  -> nyak201 @ 192.168.0.201:9000 [RETURN]
26
27    SHARE = [DEPLOY_200]  -> nyak200 @ 192.168.0.200:9000 [DEPLOY]
28    SHARE = [DEPLOY_201]  -> nyak201 @ 192.168.0.201:9000 [DEPLOY]
```

```
29    SHARE = [RETURN_200]  -> nyak200 @ 192.168.0.200:9000 [RETURN]
30    SHARE = [RETURN_201]  -> nyak201 @ 192.168.0.201:9000 [RETURN]
31  }
```

The last piece of the command and control process started with uTermCommand is implemented on the vehicle within the autonomy module, pHelmIvP. One may configure the helm behaviors to all have as a precondition `DEPLOY=true` and also have a way-point behavior configured to a convenient return point with the precondition `RETURN=true`.

5.5 Connecting uTermCommand to the MOOSDB Under an Alias

A convention of MOOS is that each application connected to the MOOSDB must register with a unique name. Typically the name used by a process to regiister with the MOOSDB is the process name, e.g., `uTermCommand`. One may want to run multiple instances of `uTermCommand` all connected to the same MOOSDB. To support this, an optional command line argument may be provided when launching `uTermCommand`:

```
> uTermCommand file.moos --alias=uTermCommandAlpha
```

The command line argument may also be invoked from within `pAntler` to launch multiple `uTermCommand` instances simultaneously.

5.6 Publications and Subscriptions for uTermCommand

5.6.1 Variables Published by the uTermCommand Application

- `USER-DEFINED`: The only variables published are those that are poked. These variables are specified in the MOOS configuration block as described in Section 5.2.

5.6.2 Variables Subscribed for by the uTermCommand Application

- `None`

6 pEchoVar: Re-publishing Variables Under a Different Name

The pEchoVar application is a lightweight process that runs without any user interaction for "echoing" the posting of specified variable-value pairs with a follow-on posting having different variable name. For example the posting of `FOO = 5.5` could be echoed such that `BAR = 5.5` immediately follows the first posting. The motivation for developing this tool was to mimic the capability of pNav (see the MOOS website) for passing through sensor values such as `GPS_X` to become `NAV_X`. More on this in Section 6.3.

6.1 Overview of the pEchoVar Interface and Configuration Options

The `pEchoVar` application may be configured with a configuration block within a `.moos` file. 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.

6.1.1 Brief Summary of the pEchoVar Configuration Parameters

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

`ECHO:`	A mapping from one MOOS variable to another constituting an echo.
`FLIP:`	A description of how components from one variable are re-posted under another MOOS variable.
`CONDITION:`	A logic condition that must be met or all echo and flip publications are held.
`HOLD_MESSAGES:`	If `true`, messages are held when conditions are not met for later processing when logic conditions are indeed met (true).

6.1.2 MOOS Variables Posted by pEchoVar

The primary output of `pEchoVar` to the MOOSDB is the set of configured postings from the echo and flip mappings. One other variable is published on each iteration:

`PEV_ITER:`	The iteration counter for the pEchoVar Iterate() loop.
`<user-defined>:`	Any MOOS variable specified in either the `ECHO` or `FLIP` config parameters.

6.1.3 MOOS Variables Subscribed for by pEchoVar

The `pEchoVar` application will subscribe for any MOOS variables found in the antecedent of either the `echo` or `flip` mappings. It will also subscribe for any MOOS variable found in any of its logic conditions.

6.2 Basic Usage of the pEchoVar Utility

Configuring `pEchoVar` minimally involves the specification of one or more `echo` or `flip` mapping events. It may also optionally involve specifying one or more logic conditions that must be met before mapping events are posted.

6.2.1 Configuring Echo Mapping Events

An *echo event mapping* maps one MOOS variable to another. Each mapping requires one line of the form:

```
Echo = <source_variable> -> <target_variable>
```

The `<source_variable>` and `<target_variable>` components are case sensitive since they are MOOS variables. A source variable can be echoed to more than one target variable. If the set of lines forms a cycle, this will be detected and `pEchoVar` will quit with an error message indicating the cycle detection. It will also write this error message in the `.ylog` file if the `pLogger` logging tool is running. An example configuration is given in Listing 22.

Listing 22 - An example pEchoVar configuration block.

```
1  //---------------------------------------------------------
2  // pEchoVar configuration block
3
4  ProcessConfig = pEchoVar
5  {
6    AppTick   = 20
7    CommsTick = 20
8
9    Echo = GPS_X           ->  NAV_X
10   Echo = GPS_Y           ->  NAV_Y
11   Echo = COMPASS_HEADING ->  NAV_HEADING
12   Echo = GPS_SPEED       ->  NAV_SPEED
13 }
```

6.2.2 Configuring Flip Mapping Events

The `pEchoVar` application can be used to "flip" a variable rather then doing a simple echo. A flipped variable, like an echoed variable, is one that is republished under a different name, but a flipped variable parses the contents of a string comprised of a series of `'='`-separated pairs, and republishes a portion of the series under the new variable name. For example, the following string,

```
ALPHA = "xpos=23, ypos=-49, depth=20, age=19.3, certainty=1"
```

may be flipped to publish the below new string, with the fields `xpos`, `ypos`, and `depth` replaced with `x`, `y`, `vehicle_depth` respectively.

```
BRAVO = "x=23, y=-49, vehicle_depth=20"
```

The above "flip relationship", and each such relationship, is configured with the following form:

```
FLIP:<key>   = source_variable  = <variable>
FLIP:<key>   = dest_variable    = <variable>
FLIP:<key>   = source_separator = <separator>
FLIP:<key>   = dest_separator   = <separator>
FLIP:<key>   = filter           = <variable>=<value>
FLIP:<key>   = <old-field> -> <new-field>
FLIP:<key>   = <old-field> -> <new-field>
```

The relationship is distinguished with a <key>, and several components. The source_variable and dest_variable components are mandatory and must be different. The source_separator and dest_separator components are optional with default values being the string ",". Fields in the source variable will only be included in the destination variable if they are specified in a component mapping <old-vield> -> <new-field>. The example configuration in Listing 23 implements the above described example flip mapping. In this case only postings that satisfy the further filter, certainty=1, will be posted.

Listing 23 - An example pEchoVar configuration block with flip mappings.

```
1  //------------------------------------------------------------
2  // pEchoVar configuration block
3
4  ProcessConfig = pEchoVar
5  {
6    AppTick   = 20
7    CommsTick = 20
8
9    FLIP:1   = source_variable  = ALPHA
10   FLIP:1   = dest_variable    = BRAVO
11   FLIP:1   = source_separator = ,
12   FLIP:1   = dest_separator   = ,
13   FLIP:1   = filter = certainty=1
14   FLIP:1   = ypos -> y
15   FLIP:1   = xpos -> x
16 }
```

Some caution should be noted with flip mappings - the current implementation does not check for loops, as is done with echo mappings.

6.2.3 Applying Conditions to the Echo and Flip Operation

The execution of the mappings configured in pEchoVar may be configured to depend on one or more logic conditions. If conditions are specified in the configuration block, all specified logic conditions must be met or else the posting of echo and flip mappings will be suspended. The logic conditions are configured as follows:

```
CONDITION = <logic-expression>
```

The <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.

If conditions are specified, pEchoVar will automatically subscribe to all MOOS variables used in the condition expressions. If the conditions are not met, all incoming mail messages that would otherwise result in an echo of flip posting, are held. When or if the conditions are met at some point later, those mail messages are processed in the order received and echo and flip mappings may be posted en masse. However, if several mail messages for a given MOOS variable are received and stored while conditions are unmet, only the latest received mail message for that variable will be processed. As an example, consider pEchoVar configured with the below two lines:

```
ECHO      = FOO -> BAR
CONDITION = DEGREES <= 32
```

If the condition is not met for some period of time, and the following mail were received during this interval: `FOO="apples", FOO="pears", FOO="grapes"`, followed by `DEGREES=30`, then `pEchoVar` would immediately post `BAR="grapes"` immediately on the very iteration that the `DEGREES=30` message was received. Note that `BAR="apples"` and `BAR="pears"` would never be posted.

The user may alternatively configure `pEchoVar` to *not* hold incoming mail messages when or if it is in a state where its logic conditions are not met. This can be done by setting

```
HOLD_MESSAGES = false   // The default is true
```

When configured this way, upon meeting the specified logic conditions, `pEchoVar` will begin processing echo and flip mappings when or if new mail messages are received relevant to the mappings. In the above example, once `DEGREES=30` is received by `pEchoVar`, nothing would be posted until new incoming mail on the variable `FOO` is received (not even `BAR="grapes"`).

6.3 Configuring for Vehicle Simulation with pEchoVar

When in simulation mode with uSimMarine, the navigation information is generated by the simulator and not the sensors such as GPS or compass as indicated in lines 9-12 in Listing 22. The simulator instead produces `USM_*` values which can be echoed as `NAV_*` values as shown in Listing 24.

Listing 24 - An example pEchoVar configuration block during simulation.

```
1  //--------------------------------------------------------------
2  // pEchoVar configuration block (for simulation mode)
3
4  ProcessConfig = pEchoVar
5  {
6    AppTick   = 20
7    CommsTick = 20
8
9    Echo = USM_X       ->  NAV_X
10   Echo = USM_Y       ->  NAV_Y
11   Echo = USM_HEADING ->  NAV_HEADING
12   Echo = USM_SPEED   ->  NAV_SPEED
13 }
```

7 uProcessWatch: Monitoring Process Connections to the MOOSDB

7.1 Brief Overview

The uProcessWatch application is process for monitoring the presence of other MOOS processes, identified through the uProcessWatch configuration, to be present and connected to the MOOSDB under normal healthy operation. It does output a health report to the terminal, but typically is running without any terminal or GUI being display. The health report is summarized in two MOOS variables - `PROC_WATCH_SUMMARY` and `PROC_WATCH_EVENT`. The former is either set to "All Present" as in line 3 of Listing 13, or is composed of a comma-separated list of missing processes identified as being AWOL (absent without leave), as shown in line 3 of Listing 15. The `PROC_WATCH_EVENT` variable lists the last event affecting the summary, such as the re-emergence of an AWOL process or the disconnection of a process and thus new member on the AWOL list.

7.2 Configuration Parameters for uProcessWatch

Configuration of uProcessWatch is done by declaring a *watch list* in the uProcessWatch configuration block of the MOOS file. Each process to be monitored is identified on a separate line of the form:

```
WATCH = process_name[*]
```

The *process_name* field is case sensitive. The asterisk is optional and affects the strictness in pattern matching the process to the list of known healthy processes. Duplicates, should they be erroneously listed twice, are simply ignored. An example configuration is given in Listing 25.

Listing 25 - An example uProcessWatch configuration block.

```
1  //------------------------------------------------------------
2  // uProcessWatch configuration block
3
4  ProcessConfig = uProcessWatch
5  {
6    AppTick  = 2
7    CommsTick = 2
8
9    // Declare the watch-list below
10   WATCH  = pHelmIvP
11   WATCH  = uSimMarine
12   WATCH  = pEchoVar
13   WATCH  = pLogger
14   WATCH  = pMarinePID
15   WATCH  = pNodeReporter
16   WATCH  = pMOOSBridge*
17 }
```

Monitoring the state of items on the watch list is done by examining the contents of the variable `DB_CLIENTS` which is a comma separated list of clients connected to the MOOSDB. A strict pattern match is done between an item on the watch list and members of `DB_CLIENTS`. The optional asterisk after the process name indicates that a looser pattern match is performed. This is to accommodate MOOS processes that may have their names chosen at run time, such as uXMS, or may have suffixes

related to their community name such as pMOOSBridge. The `WATCH = pMOOSBridge*` declaration on line 16 in Listing 25 would not report this process as AWOL even if it is connected to the MOOSDB as `pMOOSBridge_alpha`.

7.3 Publications and Subscriptions for uProcessWatch

Variables published by the uProcessWatch application

- `PROC_WATCH_SUMMARY`: A string set either to "All Present" as in line 3 of Listing 13, or composed of a comma-separated list of missing processes identified as being AWOL (absent without leave), as shown in line 3 of Listing 15.
- `PROC_WATCH_EVENT`: A string containing the last event affecting the summary, such as the re-emergence of an AWOL process or the disconnection of a process and thus new member on the AWOL list.

Variables subscribed for by the uProcessWatch application

- `DB_CLIENTS`: Published by the MOOSDB, this variable contains a comma-separated list of processes currently connected to the MOOSDB. This list is what uProcessWatch scans and checks for missing processes.

8 uPokeDB: Poking the MOOSDB from the Command Line

8.1 Brief Overview

The uPokeDB application is a lightweight process that runs without any user interaction for writing to (poking) a running MOOSDB with one or more variable-value pairs. It is run from a console window with no GUI. It accepts the variable-value pairs from the command line, connects to the MOOSDB, displays the variable values prior to poking, performs the poke, displays the variable values after poking, and then disconnects from the MOOSDB and terminates. It also accepts a .moos file as a command line argument to grab the IP and port information to find the MOOSDB for connecting. Other than that, it does not read a uPokeDB configuration block from the .moos file.

Other Methods for Poking a MOOSDB

There are few other MOOS applications capable of poking a MOOSDB. The uMS (MOOS Scope) is an application for both monitoring and poking a MOOSDB. It is substantially more feature rich than uPokeDB, and depends on the FLTK library. The iRemote application can poke the MOOSDB by using the CustomKey parameter, but is limited to the free unmapped keyboard keys, and is good when used with some planning ahead. The latest versions of uMS and iRemote are maintained on the Oxford MOOS website. The uTermCommand application (Section 5) is a tool primarily for poking the MOOSDB with a pre-defined list of variable-value pairs configured in its .moos file configuration block. The user initiates each poke by entering a keyword at a terminal window. Unlike iRemote it associates a variable-value pair with a key *word* rather than a keyboard key. The uTimerScript application (Section 9) is another tool for poking the MOOSDB with a pre-defined list of variable-value pairs configured in its .moos file configuration block. Unlike uTermCommand, uTimerScript will poke the MOOSDB without requiring further user action, but instead executes its pokes based on a timed script. The uMOOSPoke application, written by Matt Grund, is similar in intent to uPokeDB in that it accepts a command line variable-value pair. uPokeDB has a few additional features described below, namely multiple command-line pokes, accepting a .moos file on the command-line, and a MOOSDB summary prior and after the poke.

8.2 Command-line Arguments of uPokeDB

The command-line invocation of uPokeDB accepts two types of arguments - a .moos file, and one or more variable-value pairs. The former is optional, and if left unspecified, will infer that the machine and port number to find a running MOOSDB process is localhost and port 9000. The uPokeDB process does not otherwise look for a uPokeDB configuration block in this file. The variable-value pairs are delimited by the '=' character as in the following example:

```
uPokeDB alpha.moos FOO=bar TEMP=98.6 MOTTO="such is life" TEMP_STRING:=98.6
```

Since white-space characters on a command line delineate arguments, the use of double-quotes must be used if one wants to refer to a string value with white-space as in the third variable-value pair above. The value *type* in the variable-value pair is assumed to be a double if the value is numerical, and assumed to be a string type otherwise. If one really wants to poke with a string type that happens to be numerical, i.e., the string "98.6", then the ":=" separator must be used as in the last argument in the example above. If uPokeDB is invoked with a variable type different than that already associated with a variable in the MOOSDB, the attempted poke simply has no effect.

8.3 MOOS Poke Macro Expansion

The uPokeDB utility supports macro expansion for timestamps. This may be used to generate a proxy posting from another application that uses timestamps as part of it posting. The macro for timestamps is @MOOSTIME. This will expand to the value returned by the MOOS function call MOOSTime(). This function call is implemented to return UTC time. This following is an example:

```
$ uPokeDB file.moos FOOBAR=color=red,temp=blue,timestamp=@MOOSTIME
```

The above poke would result in a posting similar to:

```
FOOBAR = color=red,temp=blue,timestamp=10376674605.24
```

As with other pokes, if the macro is part of a posting of type double, the timestamp is treated as a double. The posting

```
$ uPokeDB file.moos TIME_OF_START=@MOOSTIME
```

would result in the posting of type double for the variable TIME_OF_START, assuming it has been posted previously as a different type.

8.4 Command Line Specification of the MOOSDB to be Poked

The specification of a MOOS file on the command line is optional. The only two pieces of information uPokeDB 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:

```
$ uPokeDB FOO=bar server_host=18.38.2.158  server_port=9000
```

If the server_host or the server_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 is when the MOOSDB is running on the local machine ("localhost") with port 9000, these are the default values and the user can simply hit the return key.

```
$ uPokeDB FOO=bar                       // User launches with no info on server host or port
$ Enter Server: [localhost]             // User accepts the default by just hitting Return key
$    The server is set to "localhost"   // Server host is 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 is confirmed to be set to "9123"
```

8.5 Session Output from uPokeDB

The output in Listing 26 shows an example session when a running MOOSDB is poked with the following invocation:

```
uPokeDB alpha.moos DEPLOY=true RETURN=true
```

Lines 1-18 are standard output of a MOOS application that has successfully connected to a running MOOSDB. Lines 20-24 indicate the value of the variables prior to being poked, along with their source, i.e., the MOOS process responsible for publishing the current value to the MOOSDB, and the time at which it was last written. The time is given in seconds elapsed since the MOOSDB was started. Lines 26-30 show the new state of the poked variables in the MOOSDB after `uPokeDB` has done its thing.

Listing 26 - An example uPokeDB session output.

```
 1  ************************************************
 2  *                                              *
 3  *       This is MOOS Client                    *
 4  *       c. P Newman 2001                       *
 5  *                                              *
 6  ************************************************
 7
 8  ---------------MOOS CONNECT----------------------
 9    contacting MOOSDB localhost:9000 -  try 00001
10    Contact Made
11    Handshaking as "uPokeDB"
12    Handshaking Complete
13    Invoking User OnConnect() callback...ok
14  ------------------------------------------------
15
16  uPokeDB AppTick   @ 5.0 Hz
17  uPokeDB CommsTick @ 5 Hz
18  uPokeDB is Running
19
20  PRIOR to Poking the MOOSDB
21    VarName               (S)ource    (T)ime      VarValue
22    ----------------      ----------  ----------   -------------
23    DEPLOY                 pHelmIvP    1.87        "false"
24    RETURN                 pHelmIvP    1.87        "false"
25
26  AFTER Poking the MOOSDB
27    VarName               (S)ource    (T)ime      VarValue
28    ----------------      ----------  ----------   -------------
29    DEPLOY                 uPokeDB     8.48        "true"
30    RETURN                 uPokeDB     8.48        "true"
```

8.6 Publications and Subscriptions for uPokeDB

Variables published by the uPokeDB application

- `USER-DEFINED`: The only variables published are those that are poked. These variables are provided on the command line. See Section 8.2.

Variables subscribed for by the uPokeDB application

- `USER-DEFINED`: Since uPokeDB provides two reports as described in the above Section 8.5, it subscribes for the same variables it is asked to poke, so it can generate its before-and-after reports.

9 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.

9.1 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.

9.1.1 Brief Summary of the uTimerScript Configuration Parameters

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 (`"unnamded"`).
`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`).

9.1.2 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:

```
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"
```

9.1.3 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 9.3.3.
`UTS_RESET`:	When received with the value of either `"true"` or `"reset"`, the timer script will be reset. Described in Section 9.2.2.
`UTS_FORWARD`:	When received with a numerical value greater than zero, the script will fast-forward by the indicated time. Described in Section 9.3.3.
`UTS_PAUSE`:	When received with the value of `"true"`, `"false"`, `"toggle"`, the script will change its pause state correspondingly. Described in Section 9.3.1.

In addition to the above MOOS variables, `uTimerScript` will subscribe for any variables involved in logic conditions, described in Section 9.3.2.

9.1.4 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 27 - Command line usage for the `uTimerScript` *tool.*

```
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
```

```
 9        Display this help message.
10    --shuffle=Boolean (true/false)
11        If true, script is recalculated on each reset.
12        If event times configured with random range, the
13        ordering may change after a reset.
14        The default is true.
15    --verbose=Boolean (true/false)
16        Display script progress and diagnostics if true.
17        The default is true.
18    --version,-v
19        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.

9.1.5 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 28 below.

Listing 28 - Example configuration of the uTimerScript application.

```
 0  ============================================================
 1  uTimerScript Example MOOS Configuration
 2  ============================================================
 3  Blue lines:     Default configuration
 4  Magenta lines:  Non-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 executing the script
46    time_warp      = 1
47  }
```

9.2 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.

9.2.1 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 9.4.

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 `SHUFFLE=true` configuration. This parameter, and resetting in general, are described in the next Section (9.2.2).

9.2.2 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 `uTimerScript`, or automatically from within `uTimerScript`. Outside-cued resets can be triggered by posting `UTS_RESET="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 = true   // Default is "false"
```

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 9.3.1. 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 9.3.1. See the example in Section 9.7.1 for a case where the `UPON_AWAKE` feature is handy.

9.3 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.

9.3.1 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=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 `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 `PAUSE_VAR` parameter in the `uTimerScript` configuration block. It has the format:

```
PAUSE_VAR = <moos-variable>    // 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`.

9.3.2 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 9.3.1, 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 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 9.7.1 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 9.3.1. 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.

9.3.3 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 = <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.

9.4 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.

9.4.1 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.

9.4.2 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=<low_value>, max=<high_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. 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.

9.4.3 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.

9.5 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.

9.5.1 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 9.7.2 for a use of random time warping to simulate random wind gusts.

9.5.2 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 `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 $2 * n$ seconds. The start delay may be specified as a single value or a range of values as below:

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

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 9.7.1 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.

9.6 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.

9.6.1 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:

```
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 9.3.1) and `conditions_ok=true` (Section 9.3.2). It has been reset twice out of a maximum of five allowed resets (`resets=2/5`, Section 9.2.2). Time warping is being deployed

(time_warp=3, Section 9.5.1), there is no start delay in use (start_delay=0, Section 9.5.2). The shuffle feature is turned off (shuffle=false, Section 9.2.2). The script is not configured to reset upon re-entering the un-paused state (awake_reset=false, Section 9.2.2).

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:

```
STATUS_VAR = <moos-variable>  // 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"
```

9.6.2 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 29.

Listing 29 - 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
 5  The Raw Script: ========================================
 6  Total Elements: 10
 7  [0] USM_FORCE_ANGLE=$[ANGLE], TIME:-1, RANGE=[0,0], POSTED=false
 8  [1] USM_FORCE_MAGNITUDE_AAD={$[MAGN]*0.2}, TIME:-1, RANGE=[2,2], POSTED=false
 9  [2] USM_FORCE_MAGNITUDE_AAD={$[MAGN]*0.2}, TIME:-1, RANGE=[4,4], POSTED=false
10  [3] USM_FORCE_MAGNITUDE_AAD={$[MAGN]*0.2}, TIME:-1, RANGE=[6,6], POSTED=false
11  [4] USM_FORCE_MAGNITUDE_AAD={$[MAGN]*0.2}, TIME:-1, RANGE=[8,8], POSTED=false
12  [5] USM_FORCE_MAGNITUDE_AAD={$[MAGN]*0.2}, TIME:-1, RANGE=[10,10], POSTED=false
13  [6] USM_FORCE_MAGNITUDE_AAD={$[MAGN]*-0.2}, TIME:-1, RANGE=[12,12], POSTED=false
14  [7] USM_FORCE_MAGNITUDE_AAD={$[MAGN]*-0.2}, TIME:-1, RANGE=[14,14], POSTED=false
15  [8] USM_FORCE_MAGNITUDE_AAD={$[MAGN]*-0.2}, TIME:-1, RANGE=[16,16], POSTED=false
16  [9] USM_FORCE_MAGNITUDE_AAD={$[MAGN]*-0.2}, TIME:-1, RANGE=[18,18], POSTED=false
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
```

```
27  [c/241.124][8.06040]: USM_FORCE_MAGNITUDE_AAD = 0.19936
28  [w/241.528][10.0767]: USM_FORCE_MAGNITUDE_AAD = 0.19936
29  [q/241.931][12.0913]: USM_FORCE_MAGNITUDE_AAD = -0.19936
30  [j/242.313][14.0038]: USM_FORCE_MAGNITUDE_AAD = -0.19936
31  [d/242.716][16.0205]: USM_FORCE_MAGNITUDE_AAD = -0.19936
32  [x/243.119][18.0340]: USM_FORCE_MAGNITUDE_AAD = -0.19936
```

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).

9.7 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.

9.7.1 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 17, 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.

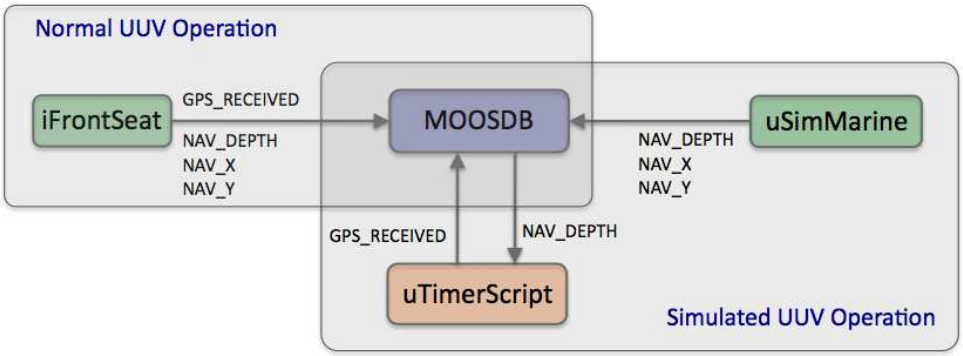


Figure 17: **Simulating a GPS Acknowledgment:** In a physical operation of the vehicle, the navigation solution and a `GPS_UPDATE_RECEIVED` heartbeat are received from the main vehicle (front-seat) computer via a MOOS module acting as an interface to the front-seat computer. In simulation, the navigation 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 30 gets the job done.

Listing 30 - 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 9.4.1 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.

9.7.2 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 18), 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.

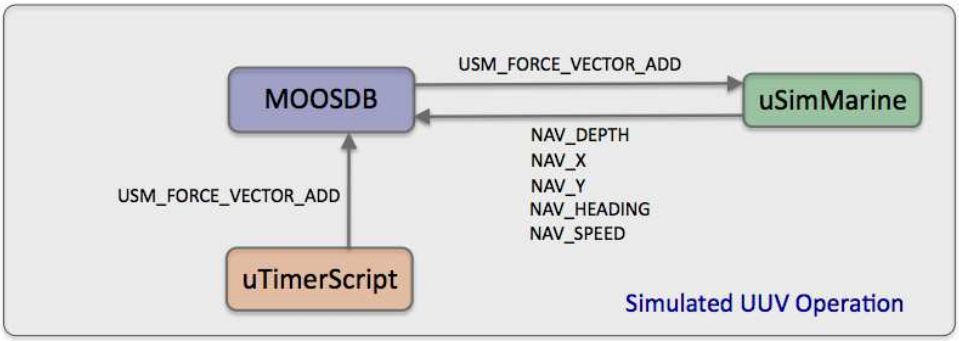


Figure 18: **Simulated Wind Gusts:** The uTimerScript application may be configured to post periodic sequences of external force values, used by the uSimMarine application to simulate wind gust effects on its simulated vehicle.

The script in Listing 31 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 31 - A uTimerScript configuration for simulating simple wind gusts.

```
0  //------------------------------------------
```

```
 1  // uTimerScript configuration  block
 2
 3  ProcessConfig = uTimerScript
 4  {
 5    AppTick   = 2
 6    CommsTick = 2
 7
 8    PAUSED        = false
 9    RESET_MAX     = unlimited
10    RESET_TIME    = end
11    DELAY_RESET   = 10:60
12    TIME_WARP     = 0.25:2.0
13    SCRIPT_NAME   = WIND
14    SCRIPT_ATOMIC = true
15
16    RANDVAR = varname=ANG, min=0,   max=359, key=at_reset
17    RANDVAR = varname=MAG, min=0.5, max=1.5, key=at_reset
18
19    EVENT = var=USM_FORCE_VECTOR_ADD, val="$[ANG],{$[MAG]*0.2}", time=0
20    EVENT = var=USM_FORCE_VECTOR_ADD, val="$[ANG],{$[MAG]*0.2}", time=2
21    EVENT = var=USM_FORCE_VECTOR_ADD, val="$[ANG],{$[MAG]*0.2}", time=4
22    EVENT = var=USM_FORCE_VECTOR_ADD, val="$[ANG],{$[MAG]*0.2}", time=6
23    EVENT = var=USM_FORCE_VECTOR_ADD, val="$[ANG],{$[MAG]*0.2}", time=8
24
25    EVENT = var=USM_FORCE_VECTOR_ADD, val="$[ANG],{$[MAG]*-0.2}", time=10
26    EVENT = var=USM_FORCE_VECTOR_ADD, val="$[ANG],{$[MAG]*-0.2}", time=12
27    EVENT = var=USM_FORCE_VECTOR_ADD, val="$[ANG],{$[MAG]*-0.2}", time=14
28    EVENT = var=USM_FORCE_VECTOR_ADD, val="$[ANG],{$[MAG]*-0.2}", time=16
29    EVENT = var=USM_FORCE_VECTOR_ADD, val="$[ANG],{$[MAG]*-0.2}", time=18
30  }
```

The force *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 force 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=end` parameter on line 10 is used to ensure that the script posts all force vectors to avoid any accumulated forces over time. The `RESET_MAX` parameter is set to `"unlimited"` to ensure the script runs indefinitely.

10 The pNodeReporter Utility: Summarizing a Node's Status

The pNodeReporter MOOS application runs on each vehicle (real or simulated) and generates node-reports (as a proxy for AIS reports) for sharing between vehicles, depicted in Figure 19. The process serves one primary function - it repeatedly gathers local platform information and navigation data and creates an AIS like report in the form of the MOOS variable NODE_REPORT_LOCAL. The NODE_REPORT messages are communicated between the vehicles and the shore or shipside command and control through an inter-MOOSDB communications process such as pMOOSBridge or via acoustic modem. Since a node or platform may both generate and receive reports, the locally generated reports are labeled with the _LOCAL suffix and bridged to the outside communities without the suffix. This is to ensure that processes running locally may easily distinguish between locally generated and externally generated reports.

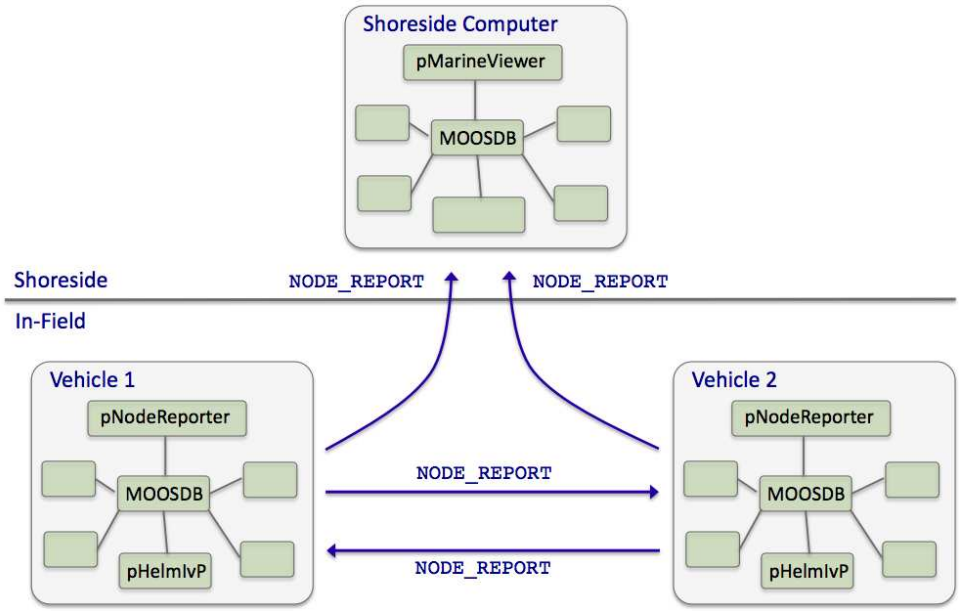


Figure 19: **Typical pNodeReporter usage:** The pNodeReporter application is typically used with pMOOSBridge or acoustic modems to share node summaries between vehicles and to a shoreside command-and-control GUI.

To generate the local report, pNodeReporter registers for the local NAV_* vehicle navigation data and creates a report in the form of a single string posted to the variable NODE_REPORT_LOCAL. An example of this variable is given in below in Section 10.1.2. The pMarineViewer and pHelmIvP applications are two modules that consume and parse the incoming NODE_REPORT messages.

The pNodeReporter utility may also publish a second report, the PLATFORM_REPORT. While the NODE_REPORT summary consists of an immutable set of data fields described later in this section, the PLATFORM_REPORT consists of data fields configured by the user and may therefore vary widely across applications. The user may also configure the frequency in which components of the PLATFORM_REPORT are posted within the report.

10.1 Overview of the pNodeReporter Interface and Configuration Options

The pNodeReporter 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 for pNodeReporter

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

ALT_NAV_PREFIX:	(10.2.5)	Source for processing alternate nav reports.
ALT_NAV_NAME:	(10.2.5)	Node name in posting alternate nav reports.
CROSS_FILL_POLICY:	(10.2.4)	Policy for handling local versus global nav reports ("literal").
BLACKOUT_INTERVAL:	(10.3)	Minimum duration, in seconds, between reports (0).
BLACKOUT_VARIANCE:	(10.3)	Variance in uniformly random blackout duration (0).
NODE_REPORT_OUTPUT:	(10.2.1)	MOOS variable used for the node report (NODE_REPORT_LOCAL).
NOHELM_THRESHOLD:	(10.2.2)	Seconds after which a quiet helm is reported as AWOL (5).
PLATFORM_LENGTH:	(10.2.3)	The reported length of the platform in meters (0).
PLATFORM_TYPE:	(10.2.3)	The reported type of the platform ("unknown").
PLAT_REPORT_OUTPUT:	(10.4)	Platform report MOOS variable (PLATFORM_REPORT_LOCAL).
PLAT_REPORT_INPUT:	(10.4)	A component of the optional platform report.

10.1.2 MOOS Variables Posted by pNodeReporter

The primary output of pNodeReporter to the MOOSDB is the node report and the optional platform report:

NODE_REPORT_LOCAL	(10.2.1)	Primary summary of the node's navigation and helm status.
PLATFORM_REPORT_LOCAL	(10.2.3)	Optional summary of certain platform characteristics.

10.1.3 MOOS Variables Subscribed for by pNodeReporter

Variables subscribed for by pNodeReporter are summarized below. A more detailed description of each variable follows. In addition to these variables, any MOOS variable that the user requests to be included in the optional PLATFORM_REPORT will also be automatically subscribed for.

IVPHELM_ENGAGED	A indicator of helm engagement produced by pHelmIvP.
IVPHELM_SUMMARY	A summary report produced by the IvP Helm (pHelmIvP).
NAV_X	The ownship vehicle position on the x axis of local coordinates.
NAV_Y	The ownship vehicle position on the y axis of local coordinates.
NAV_LAT	The ownship vehicle position on the y axis of global coordinates.
NAV_LONG	The ownship vehicle position on the x axis of global coordinates.

NAV_HEADING	The ownship vehicle heading in degrees.
NAV_YAW	The ownship vehicle yaw in radians.
NAV_SPEED	The ownship vehicle speed in meters per second.
NAV_DEPTH	The ownship vehicle depth in meters.

If pNodeReporter is configured to handle a second navigation solution as described in Section 10.2.5, the corresponding addition variables as described in that section will also be automatically subscribed for.

10.1.4 Command Line Usage of pNodeReporter

The pNodeReporter 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 pNodeReporter application is the following:

Listing 32 - Command line usage for the pNodeReporter *application.*

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

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:

```
$ pNodeReporter -e
```

This will show the output shown in Listing 33 below.

Listing 33 - Example configuration of the pNodeReporter application.

```
0  ============================================================
1  pNodeReporter Example MOOS Configuration
2  ============================================================
3  Blue lines:     Default configuration
4  Magenta lines:  Non-default configuration
5
6  ProcessConfig = pNodeReporter
7  {
8    AppTick   = 4
```

```
 9    CommsTick = 4
10
11    // Configure key aspects of the node
12    PLATFORM_TYPE      = glider   // or {uuv,auv,ship,kayak}
13    PLATFORM_LENGTH    = 8        // meters.  Range [0,inf)
14
15    // Configure optional blackout functionality
16    BLACKOUT_INTERVAL  = 0        // seconds. Range [0,inf)
17
18     // Configure the optional platform report summary
19    PLAT_REPORT_INPUT  = COMPASS_HEADING, gap=1
20    PLAT_REPORT_INPUT  = GPS_SAT, gap=5
21    PLAT_REPORT_INPUT  = WIFI_QUALITY, gap=1
22    PLAT_REPORT_OUTPUT = PLATFORM_REPORT_LOCAL
23
24    // Configure the MOOS variable containg the node report
25    NODE_REPORT_OUTPUT = NODE_REPORT_LOCAL
26
27    // Threshold for conveying an absense of the helm
28    NOHELM_THRESHOLD   = 5        // seconds
29
30    // Policy for filling in missing lat/lon from x/y or v.versa
31    CROSSFILL_POLICY   = literal  // or {fill-empty,use-latest}
32
33    // Configure monitor/reporting of dual nav solution
34    ALT_NAV_PREFIX     = NAV_GT
35    ALT_NAV_NAME       = _GT
36  }
```

10.2 Basic Usage of the pNodeReporter Utility

10.2.1 Overview Node Report Components

The primary output of pNodeReporter is the node report string. It is a comma-separated list of key-value pairs. The order of the pairs is not significant. The following is an example report:

```
NODE_REPORT_LOCAL = "NAME=alpha,TYPE=UUV,UTC_TIME=1252348077.59,X=51.71,Y=-35.50,
                     LAT=43.824981,LON=-70.329755,SPD=2.00,HDG=118.85,YAW=118.84754,
                     DEPTH=4.63,LENGTH=3.8,MODE=MODE@ACTIVE:LOITERING"
```

The UTC_TIME reflects the Coordinated Universal Time as indicated by the system clock running on the machine where the MOOSDB is running. Speed is given in meters per second, heading is in degrees in the range $[0, 360)$, depth is in meters, and the local x-y coordinates are also in meters. The source of information for these fields is the NAV_* navigation MOOS variables such as NAV_SPEED. The report also contains several components describing characteristics of the physical platform, and the state of the IvP Helm, described next.

If desired, pNodeReporter may be configured to use a different variable than NODE_REPORT_LOCAL for its node reports, with the configuration parameter NODE_REPORT_OUTPUT=FOOBAR_REPORT. Most applications that subscribe to node reports, subscribe to two variables, NODE_REPORT_LOCAL and NODE_REPORT. This is because node reports are meant to be bridged to other MOOS communities

(typically with pMOOSBridge but not necessarily). A node report should be broadcast only from the community that generated the report. In practice, to ensure that node reports that arrive in one community are not then sent out to other communities, the node reports generated locally have the _LOCAL suffix, and when they are sent to other communities they are sent to arrive with the new variable name, minus the suffix.

10.2.2 Helm Characteristics

The node report contains one field regarding the current mode of the helm, MODE. Typically the pNodeReporter and pHelmIvP applications are running on the same platform, connected to the same MOOSDB. When the helm is running, but disengaged, i.e., in manual override mode, the MODE field in the node report simply reads "MODE=DISENGAGED". When or if the helm is detected to be not running, the field reads "MODE=NOHELM-SECS", where SECS is the number of seconds since the last time pNodeReporter detected the presence of the helm, or "MODE=NOHELM-EVER" if no helm presence has ever been detected since pNodeReporter has been launched.

How does pNodeReporter know about the health or status of the helm? It subscribes to two MOOS variables published by the helm, IVPHELM_ENGAGED and IVPHELM_SUMMARY. These are described more fully in [1], but below are typical example values:

```
IVPHELM_ENGAGED  = "ENGAGED"

IVPHELM_SUMMARY  = "iter=72,ofnum=1,warnings=0,utc_time=1273494076.22,solve_time=0.00,
                   create_time=0.00,loop_time=0.00,var=course:209.0,var=speed:1.2,
                   halted=false,running_bhvs=none,modes=MODE@ACTIVE:LOITERING,
                   active_bhvs=loiter$17.8$100.00$9$0.04$0/0,completed_bhvs=none
                   idle_bhvs=waypt_return$17.8$0/0:station-keep$17.8$n/a
```

The IVPHELM_ENGAGED variable is published on each iteration of the pHelmIvP process regardless of whether the helm is in manual override ("DISENGAGED") mode or not, and regardless of whether the value of this variable has changed between iterations. It is considered the "heartbeat" of the helm. This is the variable monitored by pNodeReporter to determine whether a "NOHELM" message is warranted. By default, a period of five seconds is used as a threshold for triggering a "NOHELM" warning. This value may be changed by setting the NOHELM_THRESHOLD configuration parameter.

When the helm is indeed engaged, i.e., not in manual override mode, the value of IVPHELM_ENGAGED posting simply reads "ENGAGED", but the helm further publishes the IVPHELM_SUMMARY variable similar to the above example. If the user has chosen to configure the helm using hierarchical mode declarations (as described in [1]), the IVPHELM_SUMMARY posting will include a component such as "modes=MODE@ACTIVE:LOITERING" as above. This value is then included in the node report by pNodeReporter. If the helm is not configured with hierarchical mode declarations, the node report simply reports "MODE=ENGAGED".

10.2.3 Platform Characteristics

The node report contains three fields regarding the platform characteristics, NAME, TYPE, and LENGTH. The name of the platform is equivalent to the name of the MOOS community within which pNodeReporter is running. The MOOS community is declared as a global MOOS parameter (outside

any given process' configuration block) in the .moos mission file. The TYPE and LENGTH parameters are set in the pNodeReporter configuration block. They may alternatively derive their values from a MOOS variable posted elsewhere by another process. The user may configure pNodeReporter to use this external source by naming the MOOS variables with the PLATFORM_LENGTH_SRC and PLATFORM_TYPE_SRC parameters. If both the source and explicit values are set, as for example:

```
PLATFORM_LENGTH = 12    // meters
PLATFORM_LENGTH_SRC = SYSTEM_LENGTH
```

then the explicit length of 12 would be used only if the MOOS variable SYSTEM_LENGTH remained unwritten to by any other MOOS application connected to the MOOSDB. The platform length and type may be used by other platforms as a parameter affecting collision avoidance algorithms and protocol. They are also used in the pMarineViewer application to allow the proper platform icon to be displayed at the proper scale.

If the platform type is known, but no information about the platform length is known, certain rough default values may be used if the platform type matches one of the following: "kayak" maps to 4 meters, "uuv" maps to 4 meters, "auv" maps to 4 meters, "ship" maps to 18 meters, "glider" maps to 3 meters.

10.2.4 Dealing with Local versus Global Coordinates

A primary component of the node report is the current position of the vehicle. The pNodeReporter application subscribes for the following MOOS variables to garner this information: NAV_X, NAV_Y in local coordinates, and the pair NAV_LAT, NAV_LONG in global coordinates. These two pairs should be consistent, but what if they aren't? And what if pNodeReporter is receiving mail for one pair but not the other? Three distinct policy choices are supported:

- The default policy: node reports include exactly what is given. If NAV_X and NAV_Y are being received only, then there will be no entry in the node report for global coordinates, and vice versa. If both pairs are being received, then both pairs are reported. No attempt is made to check or ensure that they are consistent. This is the default policy, equivalent to the configuration CROSS_FILL_POLICY=literal.
- If one of the two pairs is not being received, pNodeReporter will fill in the missing pair from the other. This policy can be chosen with the configuration CROSS_FILL_POLICY=fill-empty.
- If one of the two pairs has been received more recently, the older pair is updated by converting from the other pair. The older pair may also be in a state where it has never been received. This policy can be chosen with the configuration CROSS_FILL_POLICY=fill-latest.

10.2.5 Processing Alternate Navigation Solutions

Under normal circumstances, node reports are generated reflecting the current navigation solution as defined by the incoming NAV_* variables. The pNodeReporter application can handle the case where the vehicle also publishes an alternate navigation solution, as defined by a sister set of incoming MOOS variables separate from the NAV_* variables. In this case pNodeReporter will monitor both sets of variables and may generate *two* node reports on each iteration. The following two configuration parameters are needed to activate this capability:

```
ALT_NAV_PREFIX = <prefix>      // example: NAV_GT_
ALT_NAV_NAME   = <node-name>   // example: _GT
```

The configuration parameter, ALT_NAV_PREFIX, names a prefix for the alternate incoming navigation variables. For example, ALT_NAV_PREFIX=NAV_GT_ would result in pNodeReporter subscribing for NAV_GT_X, NAV_GT_Y and so on. A separate vehicle state would be maintained internally based on this alternate set of navigation information and a second node report would be generated.

A second node report would be published under the same MOOS variable, NODE_REPORT_LOCAL, but the NAME component of the report would be distict base on the value provided in the ALT_NAV_NAME parameter. If a name is provided that does not begin with an underscore character, that name is used. If the name does begin with an underscore, the name used in the report is the otherwise configured name of the vehicle plus the suffix.

10.3 The Optional Blackout Interval Option

Under normal circumstances, the pNodeReporter application will post a node report once per iteration, the gap between postings being determined solely by the APP_TICK parameter (Figure 20). However, there are times when it is desirable to add an artificial delay between postings. Node reports are typically only useful as information sent to another node, or to a shoreside computer rendering fielded vehicles, and there are often dropped node report messages due to the uncertain nature of communications in the field, whether it be acoustic communications, wifi, or satellite link.

Applications receiving node reports usually implement provisions that take dropped messages into account. A collision-avoidance or formation-following behavior, or a contact manager, may extrapolate a contact position from its last received position and trajectory. A shoreside command-and-control GUI such as pMarineViewer may render an interpolation of vehicle positions between node reports. To test the robustness of applications needing to deal with dropped messages, a way of simulating the dropped messages is desired. One way is to add this to the simulation version of whatever communications medium is being used. For example, there is an acoustic communications simulator where the dropping of messages may be simulated, where the probability of a drop may even be tied to the range between vehicles. Another way is to simply simulate the dropped message at the source, by adding delay to the posting of reports by pNodeReporter.

By setting the BLACKOUT_INTERVAL parameter, pNodeReporter may be configured to ensure that a node report is not posted until *at least* the duration specified by this parameter has elapsed, as shown in Figure 21.

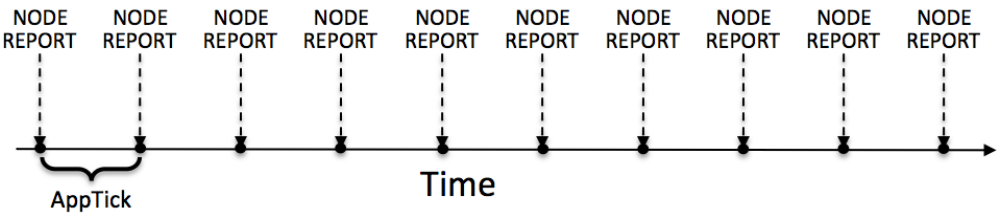


Figure 20: **Normal schedule of node report postings:** The pNodeReporter application will post node reports once per application iteration. The duration of time between postings is directly tied to the frequency at which pNodeReporter is configured to run, as set by the standard MOOS AppTick parameter.

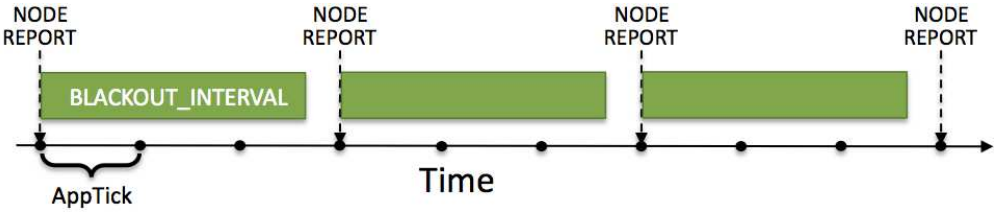


Figure 21: **The optional blackout interval parameter:** The schedule of node report postings may be altered by the setting the `BLACKOUT_INTERVAL` parameter. Reports will not be posted until at least the time specified by the blackout interval has elapsed since the previous posting.

An element of unpredictability may be added by specifying a value for the `BLACKOUT_VARIANCE` parameter. This parameter is given in seconds and defines an interval $[-t, t]$ from which a value is chosen with uniform probability, to be added to the duration of the blackout interval. This variation is re-calculated after each interval determination. The idea is depicted in Figure 22.

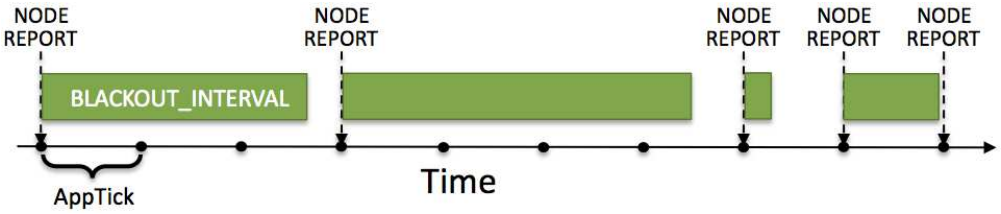


Figure 22: **Blackout intervals with varying duration:** The duration of a blackout interval may be configured to vary randomly within a user-specified range, specified in the `BLACKOUT_VARIANCE` parameter.

Message dropping is typically tied semi-predictably to characteristics of the environment, such as range between nodes, water temperature or platform depth, an so on. This method of simulating dropped messages captures none of that. It is however simple and allows for easily proceeding with the testing of applications that need to deal with the dropped messages.

10.4 The Optional Platform Report Feature

The `pNodeReporter` application allows for the optional reporting of another user-specified list of information. This report is made by posting the `PLATFORM_REPORT_LOCAL` variable. An alternative variable name may be used by setting the `PLAT_REPORT_SUMMARY` configuration parameter. This report may be configured by specifying one or more components in the `pNodeReporter` configuration block, of the following form:

```
PLAT_REPORT_INPUT = <variable>, gap=<duration>, alias=<variable>
```

If no component is specified, then no platform report will be posted. The `<variable>` element specifies the name of a MOOS variable. This variable will be automatically subscribed for by `pNodeReporter` and included in (not necessarily all) postings of the platform report. If the variable `BODY_TEMP` is specified, a component of the report may contain `"BODY_TEMP=98.6"`. An alias for a

MOOS variable may be specified. For example, alias=T, for the BODY_TEMP component would result in "T=98.6" in the platform report instead.

How often is the platform report posted? Certainly it will not be posted any more often than the AppTick parameter allows, but it may be posted far more infrequently depending on the user configuration and how often the values of its components are changing. The platform report is posted only when one or more of its components requires a re-posting. A component requires a re-posting only if (a) its value has changed, *and* (b) the time specified by its gap setting has elapsed since the last platform report that included that component. When a PLATFORM_REPORT_LOCAL posting is made, only components that required a posting will be included in the report.

The wide variation in configurations of the platform report allow for reporting information about the node that may be very specific to the platform, not suitable for a general-purpose node report. As an example, consider a situation where a shoreside application is running to monitor the platform's battery level and whether or not the payload compartment has suffered a breach, i.e., the presence of water is detected inside. A platform report could be configured as follows:

```
PLAT_REPORT_INPUT = ACME_BATT_LEVEL, gap=300, alias=BATTERY_LEVEL
PLAT_REPORT_INPUT = PAYLOAD_BREACH
```

This would result in an initial posting of:

```
PLATFORM_REPORT_LOCAL = "platform=alpha,utc_time=1273510720.99,BATTERY_LEVEL=97.3,
                        PAYLOAD_BREACH=false"
```

In this case, the platform uses batteries made by the ACME Battery Company and the interface to the battery monitor happens to publish its value in the variable ACME_BATT_LEVEL, and the software on the shoreside that monitors all vehicles in the field accepts the generic variable BATTERY_LEVEL, so the alias is used. It is also known that the ACME battery monitor output tends to fluctuate a percentage point or two on each posting, so the platform report is configured to include a battery level component no more than once every five minutes, (gap=300). The MOOS process monitoring the indication of a payload breach is known to have few false alarms and to publish its findings in the variable PAYLOAD_BREACH. Unlike the battery level which has frequent minor fluctuations and degrades slowly, the detection of a payload breach amounts to the flipping of a Boolean value and needs to be conveyed to the shoreside as quickly as possible. Setting gap=0, the default, ensures that a platform report is posted on the very next iteration of pNodeReporter, presumably to be read by a MOOS process controlling the platform's outgoing communication mechanism.

10.5 An Example Platform Report Configuration Block for pNodeReporter

Listing 34 below shows an example configuration block for pNodeReporter where an extensive platform report is configured to report information about the autonomous kayak platform to support a "kayak dashboard" display running on a shoreside computer. Most of the components in the platform report are specific to the autonomous kayak platform, which is precisely why this information is included in the platform report, and not the node report.

Listing 34 - An example pNodeReporter configuration block.

```
0  //------------------------------------------------------------------
1  // pNodeReporter config block
```

```
 2
 3  ProcessConfig = pNodeReporter
 4  {
 5    AppTick = 2
 6    CommsTick = 2
 7
 8    PLATFORM_TYPE       = KAYAK
 9    PLATFORM_LENGTH     = 3.5     // Units in meters
10    NOHELM_THRESH       = 5       // The default
11    BLACKOUT_INTERVAL   = 0       // The default
12    BLACKOUT_VARIANCE   = 0       // The default
13
14    NODE_REPORT_OUTPUT = NODE_REPORT_LOCAL        // The default
15    PLAT_REPORT_OUTPUT = PLATFORM_REPORT_LOCAL    // The default
16
17    PLAT_REPORT_INPUT = COMPASS_PITCH, gap=1
18    PLAT_REPORT_INPUT = COMPASS_HEADING, gap=1
19    PLAT_REPORT_INPUT = COMPASS_ROLL, gap=1
20    PLAT_REPORT_INPUT = DB_UPTIME, gap=1
21    PLAT_REPORT_INPUT = COMPASS_TEMPERATURE, gap=1, alias=COMPASS_TEMP
22    PLAT_REPORT_INPUT = GPS_MAGNETIC_DECLINATION, gap=10, alias=MAG_DECL
23    PLAT_REPORT_INPUT = GPS_SAT, gap=5
24    PLAT_REPORT_INPUT = DESIRED_RUDDER, gap=0.5
25    PLAT_REPORT_INPUT = DESIRED_HEADING, gap=0.5
26    PLAT_REPORT_INPUT = DESIRED_THRUST, gap=0.5
27    PLAT_REPORT_INPUT = GPS_SPEED, gap=0.5
28    PLAT_REPORT_INPUT = DESIRED_SPEED, gap=0.5
29    PLAT_REPORT_INPUT = WIFI_QUALITY, gap=0.5
30    PLAT_REPORT_INPUT = WIFI_QUALITY, gap=1.0
31    PLAT_REPORT_INPUT = MOOS_MANUAL_OVERRIDE, gap=1.0
33
34  }
```

11 The pBasicContactMgr Utility: Managing Platform Contacts

The `pBasicContactMgr` application deals with information about other known vehicles in its vicinity. It is not a sensor application, but rather handles incoming "contact reports" which may represent information received by the vehicle over a communications link, or may be the result of on-board sensor processing. By default the `pBasicContactMgr` posts to the MOOSDB summary reports about known contacts, but it also may be configured to post alerts, i.e., MOOS variables, with select content about one or more of the contacts.

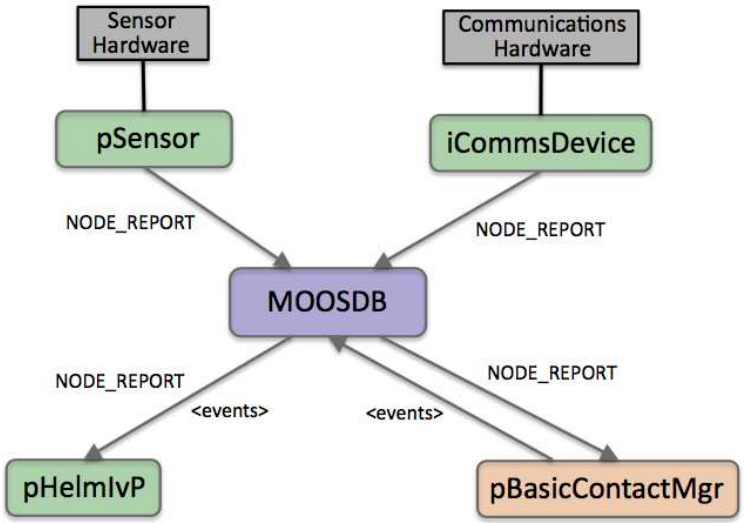


Figure 23: **The pBasicContactMgr Application:** The `pBasicContactMgr` utility receives `NODE_REPORT` information from other MOOS applications and manages a list of unique contact records. It may post additional user-configurable alerts to the MOOSDB based on the contact information and user-configurable conditions. The source of contact information may be external (via communications) or internal (via on-board sensor processing). The pSensor and iCommsDevice modules shown here are fictional applications meant to convey these two sources of information abstractly.

The `pBasicContactMgr` application is partly designed with simultaneous usage of the IvP Helm in mind. The alerts posted by `pBasicContactMgr` may be configured to trigger the dynamic spawning of behaviors in the helm, such as collision-avoidance behaviors. The `pBasicContactMgr` application does not perform sensor fusion, and does not reason about or post information regarding the confidence it has in the reported contact position relative to ground truth. These may be features added in the future, or perhaps may be features of an alternative contact manager application developed by a third party source.

11.1 Overview of the pBasicContactMgr Interface and Configuration Options

The `pBasicContactMgr` 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.

11.1.1 Brief Summary of the pBasicContactMgr Configuration Parameters

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

ALERT:	A description of a single alert.
ALERT_RANGE:	The range to a contact, in meters, within which an alert is posted (1,000).
ALERT_CPA_RANGE:	The range to a contact, in meters, within which an alert is posted if CPA over ALERT_CPA_TIME falls within the ALERT_RANGE distance (1,000).
ALERT_CPA_TIME:	The time, in seconds, for which ALERT_CPA_RANGE is calculated (0).
CONTACT_MAX_AGE:	Seconds between reports before a contact is dropped from the list (3600).
DISPLAY_RADII:	If true, the two alert ranges are posted as viewable circles (false).
VERBOSE:	If true, progress output is generated to the console (true).

11.1.2 MOOS Variables Posted by pBasicContactMgr

The primary output of pBasicContactMgr to the MOOSDB is the set of user-configured alerts. Other variables are published on each iteration where a change is detected on its value:

CONTACTS_LIST:	A comma-separated list of contacts.
CONTACTS_RECAP:	A comma-separated list of contact summaries.
CONTACTS_ALERTED:	A list of contacts for which alerts have been posted.
CONTACTS_UNALERTED:	A list of contacts for which alerts are pending, based on the range criteria.
CONTACTS_RETIRED:	A list of contacts removed due to the information staleness.
CONTACT_MGR_WARNING:	A warning message indicating possible mishandling of or missing data.

Some examples:

```
CONTACTS_LIST = "delta,gus,charlie,henry"
CONTACTS_ALERTED = "delta,charlie"
CONTACTS_UNALERTED = "gus,henry"
CONTACTS_RETIRED = "bravo,foxtrot,kilroy"
CONTACTS_RECAP = "name=delta,age=11.3,range=193.1 # name=gus,age=0.7,range=48.2 #  \
                 name=charlie,age=1.9,range=73.1 # name=henry,age=4.0,range=18.2"
```

11.1.3 MOOS Variables Subscribed for by pBasicContactMgr

The pBasicContactMgr application will subscribe for the following MOOS variables:

CONTACT_RESOLVED:	A name of a contact that has been declared resolved.
NODE_REPORT:	A report about a known contact.
NAV_X:	Present position of ownship in local x coordinates.
NAV_Y:	Present position of ownship in local y coordinates.
NAV_HEADING:	Present ownship heading in degrees.

NAV_SPEED: Present ownship speed in meters per second.

NAV_DEPTH: Present ownship depth in meters.

11.1.4 Command Line Usage of pBasicContactMgr

The pBasicContactMgr 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 pBasicContactMgr application is the following:

Listing 35 - Command line usage for the pBasicContactMgr *application.*

```
 0  Usage: pBasicContactMgr file.moos [OPTIONS]
 1
 2  Options:
 3    --alias=<ProcessName>
 4        Launch pBasicContactMgr with the given process name
 5        rather than pBasicContactMgr.
 6    --example, -e
 7        Display example MOOS configuration block.
 8    --help, -h
 9        Display this help message.
10    --verbose=<Boolean>
11        Display status updates and diagnostics if true.
12        The default is true.
13    --version,-v
14        Display the release version of pBasicContactMgr.
```

11.1.5 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:

```
$ pBasicContactMgr -e
```

This will show the output shown in Listing 36 below.

Listing 36 - Example configuration of the pBasicContactMgr application.

```
 0  ============================================================
 1  pBasicContactMgr Example MOOS Configuration
 2  ============================================================
 3  Blue lines:     Default configuration
 4  Magenta lines:  Non-default configuration
 5
 6  ProcessConfig = pBasicContactMgr
 7  {
 8    AppTick   = 4
 9    CommsTick = 4
10
11    // Alert configurations (one or more)
12    alert = var=CONTACT_INFO, val="name=avd_$[VNAME] # contact=$[VNAME]"
```

```
13
14    // Properties for all alerts
15    alert_range      = 1000      // meters.   Range [0,inf)
16    alert_cpa_range  = 1000      // meters.   Range [0,inf)
17    alert_cpa_time   = 0         // seconds.  Range [0,inf)
18
19    // Policy for retaining potentiall stale contacts
20    contact_max_age  = 3600      // seconds.  Range [0,inf)
21
22    // Configuring other output
23    display_radii    = false     // or {true}
24    verbose          = true      // or {false}
25  }
```

11.2 Basic Usage of the pBasicContactMgr Utility

The operation of `pBasicContactMgr` consists of posting user-configured alerts, and the posting of several MOOS variables, the `CONTACTS_*` variables, indicating the status of the contact manager.

11.2.1 Contact Alert Messages

Alert messages are used to alert other MOOS applications when a contact has been detected within a certain range of ownship. Messages are configured in the `pBasicContactMgr` block of the `.moos` file:

```
ALERT = var=<moos-variable>, val=<alert-content>
```

The `<alert-content>` may be any string with any, none, or all of the following macros available for expansion:

`$[VNAME]`:	The name of the contact.
`$[X]`:	The position of the contact in local x coordinates.
`$[Y]`:	The position of the contact in local y coordinates.
`$[LAT]`:	The latitude position of the contact in earth coordinates.
`$[LON]`:	The longitude position of the contact in earth coordinates.
`$[HDG]`:	The reported heading of the contact.
`$[SPD]`:	The reported speed of the contact.
`$[DEP]`:	The reported depth of the contact.
`$[VTYPE]`:	The reported vessel type of the contact.
`$[UTIME]`:	The UTC time of the last report for the contact.

The following is an example configuration:

```
ALERT = var=CONTACT_INFO, val="name=avd_$[VNAME] # contact=$[VNAME]"
```

The right-hand side of the `ALERT` specification is a #-separated list of parameter=value pairs. Note that in the above example, the value component in the `val=<alert-content>` pair itself is a string with comma-separated parameter=value pairs. Putting the whole `<alert-content>` component in double-quotes ensures that the comma separator is interpreted locally within that string.

11.2.2 Contact Alert Triggers

Alerts are triggered for all contacts based on range between ownship and the reported contact position. It is assumed that each incoming contact report minimally contains the contact's name and present position. An alert will be triggered if the current range to the contact falls within the distance given by ALERT_RANGE, as in Contact-A in Figure 24.

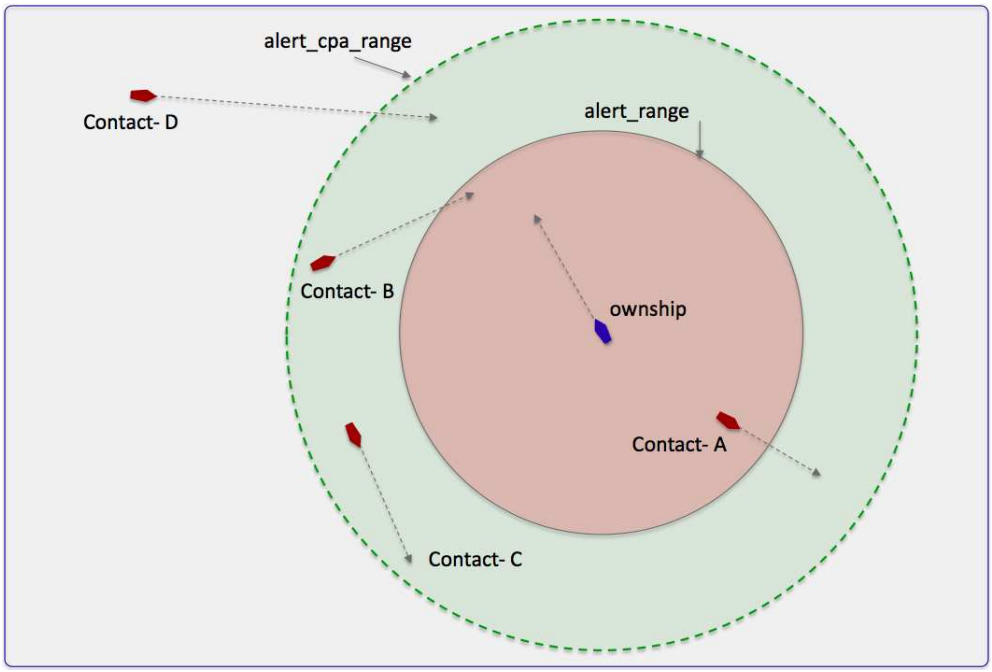


Figure 24: **Alert Triggers in pBasicContactMgr:** An alert may be triggered by pBasicContactMgr if the contact is within the alert_range, as with Contact-A. It may also be triggered if the contact is within the alert_cpa_range, and contact's CPA distance is within the alert_range, as with Contact-B. Contact-C shown here would not trigger an alert since its CPA distance is its current range and is not within the alert_range. Contact-D also would not trigger an alert despite the fact that its CPA with ownship is apparently small, since its current absolute range is outside the alert_cpa_range range.

The contact manager may also be configured with a second trigger criteria consisting of another range and time interval:

```
ALERT_CPA_RANGE = <distance>
ALERT_CPA_TIME = <duration>
```

The ALERT_CPA_RANGE is typically larger than the ALERT_RANGE. (Its influence is effectively disabled when or if it is set to be equal to or less than the ALERT_RANGE.) When a contact is outside the ALERT_RANGE, but within the ALERT_CPA_RANGE, as with Contact-B in Figure 24, the closest point of approach (CPA) between the contact and ownship is calculated given their presently-known position and trajectories. If the CPA distance falls below the ALERT_RANGE value, an alert is triggered. The ALERT_CPA_TIME interval is applied to the CPA calculation to mean that the calculated CPA distance is the CPA distance within the next ALERT_CPA_TIME seconds.

11.2.3 Contact Alert Record Keeping

The contact manager keeps a record of all known contacts for which it has received a report. This list is posted in the MOOS variable CONTACTS_LIST, in a comma-separated string such as:

```
CONTACTS_LIST = "delta,gus,charlie,henry"
```

Once an alert is generated for a contact it is put on the "alerted" list and this subset of all contacts is posted in the MOOS variable CONTACTS_ALERTED, in a comma-separated string:

```
CONTACTS_ALERTED = "delta,charlie"
```

Likewise, those contacts for which no alert has been generated are in the "unalerted" list and this is reflected in the MOOS variable CONTACTS_UNALERTED:

```
CONTACTS_UNALERTED = "gus,henry"
```

Contact records are not maintained indefinitely and eventually are "retired" from the records after some period of time during which no new reports are received for that contact. That period of time is given by the CONTACT_MAX_AGE configuration parameter. The list of retired contacts is posted in the MOOS variable CONTACTS_RETIRED:

```
CONTACTS_RETIRED = "bravo,foxtrot,kilroy"
```

A contact recap of all non-retired contacts is also posted in the MOOS variable CONTACTS_RECAP:

```
CONTACTS_RECAP = "name=delta,age=11.3,range=193.1 # name=gus,age=0.7,range=48.2 #  \
                 name=charlie,age=1.9,range=73.1 # name=henry,age=4.0,range=18.2"
```

Each of these five MOOS variables is published only when its contents differ from its previous posting.

11.2.4 Contact Resolution

An alert is generated by the contact manager for a given contact *once*, when the alert trigger criteria is first met. In the iteration when the criteria is met, the contact is moved from the "un-alerted" list to the "alerted" list, the alert is posted to the MOOSDB, and no further alerts are posted despite any future calculations of the trigger criteria. One exception to this is when the pBasicContactMgr receives notice that a contact has been "resolved", through the MOOS variable CONTACT_RESOLVED. When a contact is resolved, it is moved from the alerted list back on to the un-alerted list.

11.3 Usage of the pBasicContactMgr with the IvP Helm

The IvP helm may used in conjunction with the contact manager to coordinate the dynamic spawning of certain helm behaviors where the instance of the behavior is dedicated to a helm objective associated with a particular contact. For example, a collision avoidance behavior, or a behavior for maintaining a relative position to a contact for achieving sensing objectives, would be examples of such behaviors. One may want to arrange for a new behavior to be spawned as the

contact becomes known. The helm needs a cue in the form of a MOOS variable posting to trigger a new behavior spawning, and this is easily arranged with the alerts in the `pBasicContactMgr`.

On the flip-side of a new behavior spawning, a behavior may eventually declare itself completed and remove itself from the helm. The conditions leading to completion are defined within the behavior implementation and configuration. No cues external to the helm are required to make that happen. However, once a alert has been generated by the contact manager for a particular contact, it is not generated again, unless it receives a message that the contact has been "resolved". Therefore, if the helm wishes to received future alerts related to a contact for which it has received an alert in the past, it must declare the contact "resolved" to the contact manager as discussed in Section 11.2.4. This would be important, for example, in the following scenario: (a) a collision avoidance behavior is spawned for a new contact that has come within range, (b) the behavior completes and is removed from the helm, presumably because the contact has slipped safely out of range, (c) the contact or ownship turns such that a collision avoidance behavior is once again needed for the same contact.

An example mission is available for showing the use of the contact manager and its coordination with the helm to spawn behaviors for collision avoidance. This mission is `m2_berta` and is described in the IvP Helm documentation. In this mission two vehicles are configured to repeatedly go in and out of collision avoidance range, and the contact manager repeatedly posts alerts that result in the spawning of a collision avoidance behavior in the helm. Each time the vehicle goes out of range, the behavior completes and dies off from the helm and is declared to the contact manager to be resolved.

11.4 Console Output Generated by pBasicContactMgr

The status of the contact manager may be monitored from from an open console window where pBasicContactMgr is launched, if the verbose setting is turned on (by default). Example output is shown below in Listing 37.

Listing 37 - Example pBasicContactMgr console output.

```
 0  -------------------- Iteration: 407
 1   Time: 202.563
 2         List: vehicle1
 3      Alerted: vehicle1
 4    UnAlerted:
 5      Retired:
 6        Recap: vname=vehicle1,range=34.36,age=1.26
 7
 8  Recent Alerts:
 9    [0.00]: CONTACT_INFO=name=avd_vehicle1#contact=vehicle1
10    [81.27]: Resolved: vehicle1
11    [133.09]: CONTACT_INFO=name=avd_vehicle1#contact=vehicle1
```

In lines 2-6, the record-keeping status of the contact manager is output. These five lines are equivalent to the content of the `CONTACTS_*` variables described in Section 11.2.3. The iteration number on line 0 is the iteration counter associated with the Iterate() loop of `pBasicContactMgr`. The time stamp on line 1 represents the duration of time since the `pBasicContactMgr` was launched.

The Recent Alerts output in lines 8-11 reflect the ten most recent alert related events - either an actual alert being posted or a contact resolution.

12 The uSimMarine Utility: Basic Vehicle Simulation

The uSimMarine application is a simple 3D vehicle simulator that updates vehicle state, position and trajectory, based on the present actuator values and prior vehicle state. The typical usage scenario has a single instance of uSimMarine associated with each simulated vehicle, as shown in Figure 25.

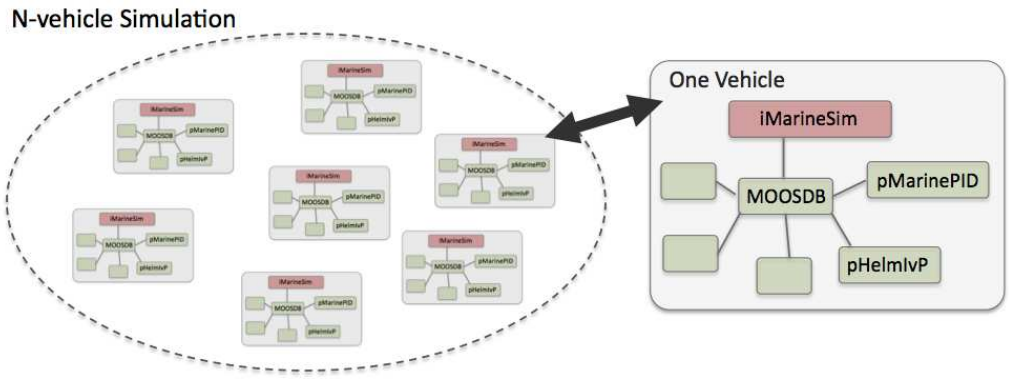


Figure 25: **Typical uSimMarine Usage:** In an N-vehicle simulation, an instance if uSimMarine is used for each vehicle. Each simulated vehicle typically has its own dedicated MOOS community. The IvP Helm (pHelmIvP) publishes high-level control decisions. The PID controller (pMarinePID) converts the high-level control decisions to low-level actuator decisions. Finally the simulator (uSimMarine) reads the low-level actuator postings to produce a new vehicle position.

This style of simulation can be contrasted with simulators that simulate a comprehensive set of aspects of the simulation, including multiple vehicles, and aspects of the environment and communications. The uSimMarine simulator simply focuses on a single vehicle. It subscribes for the vehicle navigation state variables NAV_X, NAV_Y, NAV_SPEED, NAV_HEADING, NAV_DEPTH, as well as the actuator values DESIRED_RUDDER, DESIRED_THRUST, DESIRED_ELEVATOR. The uSimMarine accommodates a notion of external forces applied to the vehicle to crudely simulate current or wind. These forces may be set statically or may be changing dynamically by other MOOS processes. The simulator also may be configured with a simple geo-referenced data structure representing a field of water currents.

Under typical UUV payload autonomy operation, the uSimMarine and pMarinePID MOOS modules would not be present. The vehicle's native controller would handle the role of pMarinePID, and the vehicle's native navigation system (and the vehicle itself) would handle the role of uSimMarine.

12.1 Overview of the uSimMarine Interface and Configuration Options

The uSimMarine application may be configured with a configuration block within a .moos file. 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 the uSimMarine interface is provided in this section.

12.1.1 Brief Summary of the uSimMarine Configuration Parameters

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

BUOYANCY_RATE	Rate at which vehicle floats to surface at zero speed (0).
CURRENT_FIELD	A file containing the specification of a current field.
CURRENT_FIELD_ACTIVE	If `true`, simulator uses the current field if specified.
FORCE_VECTOR	A pair of external force values, direction and magnitude.
FORCE_THETA	An external rotational force in degrees per second (0).
FORCE_X	An external force value applied in the x direction (0).
FORCE_Y	An external force value applied in the y direction (0).
MAX_ACCELERATION	Maximum rate of vehicle acceleration in m/s^2 (0.5).
MAX_DECELERATION	Maximum rate of vehicle deceleration in m/s^2 (0.5).
MAX_DEPTH_RATE	Maximum rate of vehicle depth change, meters per second. (0.5).
MAX_DEPTH_RATE_SPEED	Vehicle speed at which max depth rate is achievable (2.5).
PREFIX	Prefix of MOOS variables published (`USM_`).
SIM_PAUSE	If `true`, the simulation is paused (`false`).
START_DEPTH	Initial vehicle depth in meters (0).
START_HEADING	Initial vehicle heading in degrees (0).
START_POS	A full starting position and trajectory specification.
START_SPEED	Initial vehicle speed in meters per second (0).
START_X	Initial vehicle x position in local coordinates (0).
START_Y	Initial vehicle y position in local coordinates (0).
THRUST_FACTOR	A scalar correlation between thrust and speed (20).
THRUST_MAP	A mapping between thrust and speed values.
THRUST_REFLECT	If `true`, negative thrust is simply opposite positive thrust (`false`).
TURN_LOSS	A range $[0, 1]$ affecting speed lost during a turn, (0.85).
TURN_RATE	A range $[0, 100]$ affecting vehicle turn radius, e.g., 0 is inf turn radius, (70).

12.1.2 MOOS Variables Posted by uSimMarine

The primary output of `uSimMarine` to the MOOSDB is the full specification of the updated vehicle position and trajectory, along with a few other pieces of information:

USM_DEPTH:	The updated vehicle depth in meters.
USM_FSUMMARY:	A summary of the current total external force.
USM_HEADING:	The updated vehicle heading in degrees.
USM_HEADING_OVER_GROUND:	The updated vehicle heading over ground.
USM_LAT:	The updated vehicle latitude position.
USM_LONG:	The updated vehicle longitude position.
USM_SPEED:	The updated vehicle speed in meters per second.
USM_SPEED_OVER_GROUND:	The updated speed over ground.
USM_X:	The updated vehicle x position in local coordinates.
USM_Y:	The updated vehicle y position in local coordinates.
USM_YAW:	The updated vehicle yaw in radians.

An example `USM_FSUMMARY` string: `"ang=90, mag=1.5, xmag=90, ymag=0"`.

12.1.3 MOOS Variables Subscribed for by uSimMarine

The `uSimMarine` application will subscribe for the following MOOS variables:

`DESIRED_THRUST`:	The thruster actuator setting, $[-100, 100]$.
`DESIRED_RUDDER`:	The rudder actuator setting, $[-100, 100]$.
`DESIRED_ELEVATOR`:	The depth elevator setting, $[-100, 100]$.
`USM_SIM_PAUSED`:	Simulation pause request, either `true` or `false`.
`USM_CURRENT_FIELD`:	If `true`, a configured current field is active.
`USM_BUOYANCY_RATE`:	Dynamically set the zero-speed float rate.
`USM_FORCE_THETA`:	Dynamically set the external rotational force.
`USM_FORCE_X`:	Dynamically set the external force in the x direction.
`USM_FORCE_Y`:	Dynamically set the external force in the y direction.
`USM_FORCE_VECTOR`:	Dynamically set the external force direction and magnitude.
`USM_FORCE_VECTOR_ADD`:	Dynamically modify the external force vector.
`USM_FORCE_VECTOR_MULT`:	Dynamically modify the external force vector magnitude.
`USM_RESET`:	Reset the simulator with a new position, heading, speed and depth.

Each iteration, after noting the changes in the navigation and actuator values, it posts a new set of navigation state variables in the form of `USM_X`, `USM_Y`, `USM_SPEED`, `USM_HEADING`, `USM_DEPTH`.

12.1.4 Command Line Usage of uSimMarine

The `uSimMarine` 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 `uSimMarine` application is the following:

Listing 38 - Command line usage for the `uSimMarine` *application.*

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

12.1.5 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:

```
$ uSimMarine -e
```

This will show the output shown in Listing 39 below.

Listing 39 - Example configuration of the uSimMarine application.

```
 0  ============================================================
 1  uSimMarine Example MOOS Configuration
 2  ============================================================
 3  Blue lines:     Default configuration
 4  Magenta lines:  Non-default configuration
 5
 6  ProcessConfig = uSimMarine
 7  {
 8    AppTick   = 4
 9    CommsTick = 4
10
11    start_x       = 0
12    start_y       = 0
13    start_heading = 0
14    start_speed   = 0
15    start_depth   = 0
16    start_pos     = x=0, y=0, speed=0, heading=0, depth=0
17
18    force_x       = 0
19    force_y       = 0
20    force_theta   = 0
21    force_vector  = 0,0      // heading, magnitude
22
23    buoyancy_rate        = 0.025 // meters/sec
24    max_acceleration     = 0     // meters/sec^2
25    max_deceleration     = 0.5   // meters/sec^2
26    max_depth_rate       = 0.5   // meters/sec
27    max_depth_rate_speed = 2.0   // meters/sec
28
29    sim_pause            = false // or {true}
30    dual_state           = false // or {true}
31    thrust_reflect       = false // or {true}
32    thrust_factor        = 20    // range [0,inf)
33    turn_rate            = 70    // range [0,100]
34    thrust_map           = 0:0, 20:1, 40:2, 60:3, 80:5, 100:5
35  }
```

12.2 Setting the Initial Vehicle Position, Pose and Trajectory

The simulator is typically configured with a vehicle starting position, pose and trajectory given by the following five configuration parameters:

- START_X
- START_Y
- START_HEADING
- START_SPEED

- START_DEPTH

The position is specified in local coordinates in relation to a local datum, or $(0, 0)$ position. This datum is specified in the .moos file at the global level. The heading is specified in degrees and corresponds to the direction the vehicle is pointing. The initial speed and depth by default are zero, and are often left unspecified in configuration. Alternatively, the same five parameters may be set with the START_POS parameter as follows:

```
START_POS = x=100, y=150, speed=0, heading=45, depth=0
```

The simulator can also be reset at any point during its operation, by posting to the MOOS variable USM_RESET. A posting of the following form will reset the same five parameters as above:

```
USM_RESET = x=200, y=250, speed=0.4, heading=135, depth=10
```

This has been useful in cases where the objective is to observe the behavior of a vehicle from several different starting positions, and an external MOOS script, e.g., uTimerScript, is used to reset the simulator from each of the desired starting states.

12.3 Propagating the Vehicle Speed, Heading, Position and Depth

The vehicle position is updated on each iteration of the uSimMarine application, based on (a) the previous vehicle state, (b) the elapsed time since the last update, ΔT, (c) the current actuator values, DESIRED_RUDDER, DESIRED_THRUST, and DESIRED_ELEVATOR, and (d) several parameter settings describing the vehicle model.

For simplicity, this simulator updates the vehicle speed, heading, position and depth in sequence, in this order. For example, the position is updated after the heading is updated, and the new position update is made as if the new heading were the vehicle heading for the entire ΔT. The error introduced by this simplification is mitigated by running uSimMarine with a fairly high MOOS AppTick value keeping the value of ΔT sufficiently small.

Propagating the Vehicle Speed

The vehicle speed is propagated primarily based on the current value of thrust, which is presumably refreshed upon each iteration by reading the incoming mail on the MOOS variable DESIRED_THRUST. To simulate a small speed penalty when the vehicle is conducting a turn through the water, the new thrust value may also be affected by the current rudder value, referenced by the incoming MOOS variable DESIRED_RUDDER. The newly calculated speed is also dependent on the previously noted speed noted by the incoming MOOS variable NAV_SPEED, and the settings to the two configuration parameters MAX_ACCELERATION and MAX_DECELERATION.

The algorithm for updating the vehicle speed proceeds as:

1. Calculate $v_{i(\texttt{RAW})}$, the new raw speed based on the thrust.
2. Calculate $v_{i(\texttt{TURN})}$, an adjusted and potentially lower speed, based on the raw speed, $v_{i(\texttt{RAW})}$, and the current rudder angle, DESIRED_RUDDER.

3. Calculate $v_{i(\texttt{FINAL})}$, an adjusted and potentially lower speed based on $v_{i(\texttt{TURN})}$, compared to the prior speed. If the magnitude of change violates either the max acceleration or max deceleration settings, then the new speed is clipped appropriately.
4. Set the new speed to be $v_{i(\texttt{FINAL})}$, and use this new speed in the later updates on heading, position and depth.

Step 1: In the first step, the new speed is calculated by the current value of thrust. In this case the *thrust map* is consulted, which is a mapping from possible thrust values to speed values. The thrust map is configured with the THRUST_MAP configuration parameter, and is described in detail in Section 12.5.

$$v_{i(\texttt{RAW})} = \texttt{THRUST\_MAP}(\texttt{DESIRED\_THRUST})$$

Step 2: In the second step, the calculated speed is potentially reduced depending on the degree to which the vehicle is turning, as indicated by the current value of the MOOS variable DESIRED_RUDDER. If it is not turning, it is not diminished at all. The adjusted speed value is set according to:

$$v_{i_{(\texttt{TURN})}} = v_{i(\texttt{RAW})} * (1 - (\frac{|\texttt{RUDDER}|}{100} * \texttt{TURN\_LOSS}))$$

The configuration parameter TURN_LOSS is a value in the range of $[0, 1]$. When set to zero, there is no speed lost in any turn. When set to 1, there is a 100% speed loss when there is a maximum rudder. The default value is 0.85.

Step 3: In the last step, the candidate new speed, $v_{i_{(\texttt{TURN})}}$, is compared with the incoming vehicle speed, v_{i-1}. The elapsed time since the previous simulator iteration, ΔT, is used to calculate the acceleration or deceleration implied by the new speed. If the change in speed violates either the MAX_ACCELERATION, or MAX_ACCELERATION parameters, the speed is adjusted as follows:

$$v_{i_{(\texttt{FINAL})}} = \begin{cases} v_{i-1} + (\texttt{MAX\_ACCELERATION} * \Delta T) & \frac{(v_{i_{(\texttt{TURN})}} - v_{i-1})}{\Delta T} > \texttt{MAX\_ACCELERATION}, \\ v_{i-1} - (\texttt{MAX\_DECELERATION} * \Delta T) & \frac{(v_{i-1} - v_{i_{(\texttt{TURN})}})}{\Delta T} > \texttt{MAX\_DECELERATION}, \\ v_{i_{(\texttt{TURN})}} & \text{otherwise.} \end{cases}$$

Step 4: The final speed from the previous step is posted by the simulator as USM_SPEED, and is used the calculations of position and depth, described next.

Propagating the Vehicle Heading

The vehicle heading is propagated primarily based on the current RUDDER value which is refreshed upon each iteration by reading the incoming mail on the MOOS variable DESIRED_RUDDER, and the elapsed time since the simulator previously updated the vehicle state, ΔT. The change in heading my also be influenced by the THRUST value from the MOOS variable DESIRED_THRUST, and may also factor an external rotational force.

The algorithm for updating the new vehicle heading proceeds as:

1. Calculate $\Delta\theta_{i(\texttt{RAW})}$, the new raw change in heading influenced only by the current rudder value.
2. Calculate $\Delta\theta_{i(\texttt{THRUST})}$, an adjusted change in heading, based on the raw change in heading, $\Delta\theta_{i(\texttt{RAW})}$, and the current THRUST value.
3. Calculate $\Delta\theta_{i(\texttt{EXTERNAL})}$, an adjusted change in heading considering external rotational force.
4. Calculate θ_i, the final new heading based on the calculated change in heading and the previous heading, and converted to the range of $[0, 359]$.

Step 1: In the first step, the new heading is calculated by the current RUDDER value:

$$\Delta\theta_{i(\texttt{RAW})} = \texttt{RUDDER} * \frac{\texttt{TURN\_RATE}}{100} * \Delta T$$

The TURN_RATE is an uSimMarine configuration parameter with the allowable range of $[0, 100]$. The default value of this parameter is 70, chosen in part to be consistent with the performance of the simulator prior to this parameter being exposed to configuration. A value of 0 would result in the vehicle never turning, regardless of the rudder value.

Step 2: In the second step the influence of the current vehicle thrust (from the MOOS variable DESIRED_THRUST) may be applied to the change in heading. The magnitude of the change of heading is adjusted to be greater when the thrust is greater than 50% and less when the thrust is less than 50%.

$$\Delta\theta_{i(\texttt{THRUST})} = \theta_{i(\texttt{RAW})} * (1 + \frac{|\texttt{THRUST}| - 50}{50})$$

The direction in heading change is then potentially altered based on the sign of the THRUST:

$$\Delta\theta_{i(\texttt{THRUST})} = \begin{cases} -\Delta\theta_{i(\texttt{THRUST})} & \texttt{THRUST} < 0, \\ \Delta\theta_{i(\texttt{THRUST})} & \text{otherwise.} \end{cases}$$

Step 3: In the third step, the change in heading may be further influenced by an external rotational force. This force, if present, would be read at the outset of the simulator iteration from either the configuration parameter FORCE_THETA, or dynamically from the MOOS variable USM_FORCE_THETA. The FORCE_THETA terms are a misnomer since they are expressed in degrees per second. The updated value is calculated as follows:

$$\Delta\theta_{i(\texttt{EXTERNAL})} = \theta_{i(\texttt{THRUST})} + (\texttt{FORCE\_THETA} * \Delta T)$$

Step 4: In final step, the final new heading is set based on the previous heading and the change in heading calculated in the previous three steps. If needed, the value of the new heading is converted to its equivalent heading in the range $[0, 359]$.

$$\theta_i = \texttt{heading360}(\theta_{i-1} + \Delta\theta_{i(\texttt{EXTERNAL})})$$

The simulator then posts this value to the MOOSDB as USM_HEADING.

Propagating the Vehicle Position

The vehicle position is propagated primarily based on the newly calculated vehicle heading and speed, the previous vehicle position, and the elapsed time since updating the previous vehicle position, ΔT.

The algorithm for updating the new vehicle position proceeds as:

1. Calculate the vehicle heading and speed used for updating the new vehicle position, with the heading converted into radians.
2. Calculate the new positions, x_i and y_i, based on the heading, speed and elapsed time.
3. Calculate a possibly revised new position, factoring in any external forces.

Step 1: In the first step, the heading value, $\bar{\theta}$, and speed value, $\bar{v}$ used for calculating the new vehicle position is set averaging the newly calculated values with their prior values:

$$\bar{v} = \frac{(v_i + v_{i-1})}{2} \tag{1}$$

$$\bar{\theta} = \text{atan2}(s, c)$$

where s and c are given by:

$$s = sin(\theta_{i-1}\pi/180) + sin(\theta_i\pi/180)$$
$$c = cos(\theta_{i-1}\pi/180) + cos(\theta_i\pi/180)$$

The above calculation of the heading average handles the issue of angle wrap, i.e., the average of 359 and 1 is zero, not 180.

Step 2: The vehicle x and y position is updated by the following two equations:

$$x_i = x_{i-1} + sin(\bar{\theta}) * \bar{v} * \Delta\mathtt{T}$$
$$y_i = y_{i-1} + cos(\bar{\theta}) * \bar{v} * \Delta\mathtt{T}$$

The above is calculated keeping in mind the difference in convention used in marine navigation where zero degrees is due North and 90 degrees is due East. That is, the mapping is as follows from marine to traditional trigonometric convention: $0° \rightarrow 90°$, $90° \rightarrow 0°$, $180° \rightarrow 270°$, $270° \rightarrow 180°$.

Step 3: The final step adjusts the x, and y position from above, taking into consideration any external force that may be present. This force includes both the force that may be directed from the incoming MOOS variables as described in Section 12.4. The force components below are also a misnomer since they are provided in units of meters per second.

$$x_i = x_i + \texttt{EXTERNAL\_FORCE\_X} * \Delta\mathtt{T} \tag{2}$$
$$y_i = y_i + \texttt{EXTERNAL\_FORCE\_Y} * \Delta\mathtt{T} \tag{3}$$

Propagating the Vehicle Depth

Depth change in `uSimMarine` is simulated based on a few input parameters. The primary parameter that changes from one iteration to the next is the `ELEVATOR` actuator value, from the MOOS variable `DESIRED_ELEVATOR`. On any given iteration the new vehicle depth, z_i, is determined by:

$$z_i = z_{i-1} + (\dot{z}_i * \Delta t)$$

The new vehicle depth is altered by the *depth change rate*, $\dot{z}_i$, applied to the elapsed time, Δt, which is roughly equivalent to the `AppTick` interval set in the `uSimMarine` configuration block. The depth change rate on the current iteration is determined by the vehicle speed as set in (1) and the `ELEVATOR` actuator value, and by the following three vehicle-specific simulator configuration parameters that allow for some variation in simulating the physical properties of the vehicle. The `BUOYANCY_RATE`, for simplicity, is given in meters per second where positive values represent a positively buoyant vehicle. The `MAX_DEPTH_RATE`, and `MAX_DEPTH_RATE_SPEED` parameters determine the function(s) shown in Figure 26. The vehicle will have a higher depth change rate at higher speeds, up to some maximum speed where the speed no longer affects the depth change rate. The actual depth change rate then depends on the elevator and vehicle speed.

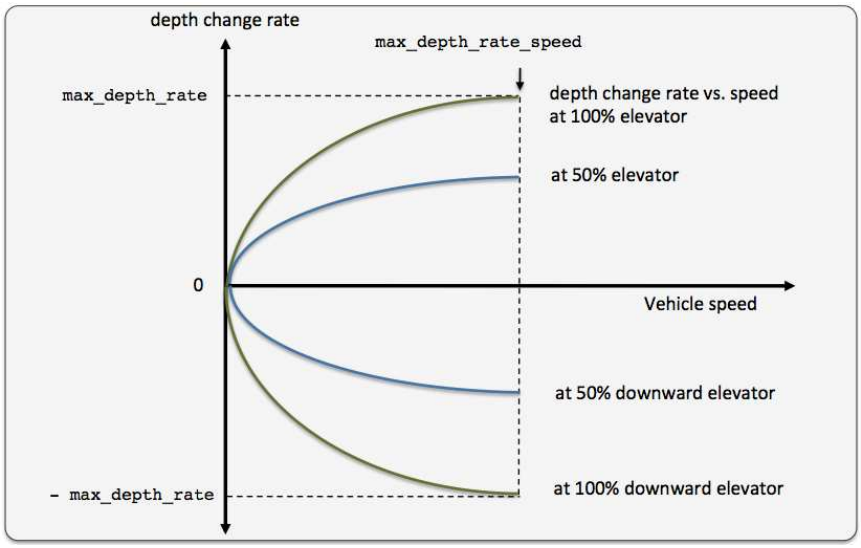


Figure 26: The relationship between the rate of depth change rate, given a current vehicle speed. Different elevator settings determine unique curves as shown.

The value of the depth change rate, $\dot{v}_i$, is determined as follows:

$$\dot{z}_i = \left(\frac{\bar{v}}{\texttt{MAX\_DEPTH\_RATE\_SPEED}}\right)^2 * \frac{\texttt{ELEVATOR}}{100} * \texttt{MAX\_DEPTH\_RATE} + \texttt{BUOYANCY\_RATE} \tag{4}$$

Both fraction components in 4 are clipped to $[-1, 1]$. When the vehicle is in reverse thrust and has a negative speed, this equation still holds. However, a vehicle would likely not have a depth

change rate curve symetric between positive and negative vehicle speeds. By default the value of BUOYANCY_RATE is set to 0.025, slightly positively buoyant, MAX_DEPTH_RATE is set to 0.5, and MAX_DEPTH_RATE_SPEED is set to 2.0. The prevailing buoyancy rate may be dynamically adjusted by a separate MOOS application publishing to the variable USM_BUOYANCY_RATE.

12.4 Simulation of External Forces

When the simulator updates the vehicle position as in equations (2) and (3), it factors a possible external force in the x and y directions, in the term EXTERNAL_FORCE_X, and EXTERNAL_FORCE_X respectively. The external force may have two distinct components; a force applied generally, and a force applied due to a current field configured with an external file correlating force vectors to local x and y positions. These forces may be set in one of three ways discussed next. Referring to these parameters in terms of *force* is an admitted misnomer, since all units are given in meters per second.

External X-Y Forces from Initial Simulator Configuration

An external force may be configured upon startup by either specifying explicitly the forces in the x and y direction, or by specifying a force magnitude and direction. Figure 27 shows two external forces each with the appropriate configuration using either the FORCE_X and FORCE_Y parameters or the single FORCE_VECTOR parameter:

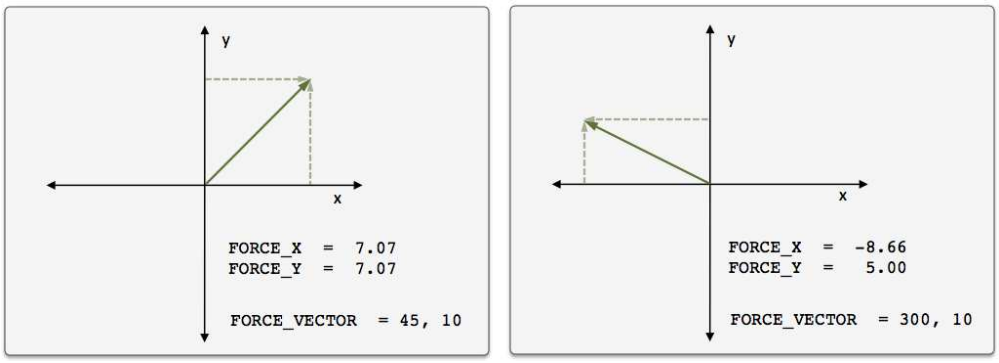


Figure 27: *External Force Vectors*: Two force vectors each configured with either the FORCE_X and FORCE_Y configuration parameters or their equivalent single FORCE_VECTOR parameter.

If, for some reason, the user mistakenly configures the simulator with both configuration styles, the configuration appearing last in the configuration block will be the prevailing configuration. If uSimMarine is configured with these parameters, these external forces will be applied on the very first iteration and all later iterations unless changed dynamically, as discussed next.

External X-Y Forces Received from Other MOOS Applications

External forces may be adjusted dynamically by other MOOS applications based on any criteria wished by the user and developer. The uSimMarine application registers for the following MOOS variables in this regard: USM_FORCE_X, USM_FORCE_Y, USM_FORCE_VECTOR, USM_FORCE_VECTOR_ADD,

USM_FORCE_VECTOR_MULT. The first three variables simply override the previously prevailing force, set by either the initial configuration or the last received mail concerning the force.

By posting to the USM_FORCE_VECTOR_MULT variable the *magnitude* of the prevailing vector may be modified with a single multiplier such as:

```
USM_FORCE_VECTOR_MULT = 2
USM_FORCE_VECTOR_MULT = -1
```

The first MOOS posting above would double the size of the prevailing force vector, and the second example would reverse the direction of the vector. The USM_FORCE_VECTOR_ADD variable describes a force vector to be *added* to the prevailing force vector. For example, consider the prevailing force vector shown on the left in Figure 27, with the following MOOS mail received by the simulator:

```
USM_FORCE_VECTOR_ADD = "262.47, 15.796"
```

The resulting force vector would be the vector shown on the right in Figure 27. This interface opens the door for the scripting changes to the force vector like the one below, that crudely simulate a gust of wind in a given direction that builds up to a certain magnitude and dies back down to a net zero force.

```
USM_FORCE_VECTOR_ADD = 137, 0.25
USM_FORCE_VECTOR_ADD = 137, 0.25
USM_FORCE_VECTOR_ADD = 137, 0.25
USM_FORCE_VECTOR_ADD = 137, 0.25
USM_FORCE_VECTOR_ADD = 137, 0.25
USM_FORCE_VECTOR_ADD = 137, -0.25
USM_FORCE_VECTOR_ADD = 137, -0.25
USM_FORCE_VECTOR_ADD = 137, -0.25
USM_FORCE_VECTOR_ADD = 137, -0.25
USM_FORCE_VECTOR_ADD = 137, -0.25
```

The above style script was described in the Section 9.7.2, where the uTimerScript utility was used to simulate wind gusts in random directions with random magnitude. The USM_FORCE_* interface may also be used by any third party MOOS application simulating things such as ocean or wind currents. The uSimMarine application does have native support for simple simulation with current fields as described next.

External X-Y Forces from a Current Field

```
CURRENT_FIELD        = gulf_of_maine.cfd
CURRENT_FIELD_ACTIVE = true

<x-position>, <y-position>, <speed>, <direction>
```

12.5 The ThrustMap Data Structure

A *thrust map* is a data structure that may be used to simulate a non-linear relationship between thrust and speed. This is configured in the `uSimMarine` configuration block with the `THRUST_MAP` parameter containing a comma-separated list of colon-separated pairs. Each element in the comma-separated list is a single mapping component. In each component, the value to the left of the colon is a thrust value, and the other value is a corresponding speed. The following is an example mapping given in string form, and rendered in Figure 28.

```
THRUST_MAP = "-100:-3.5, -75:-3.2, -10:-2, 20:2.4, 50:4.2, 80:4.8, 100:5"
```

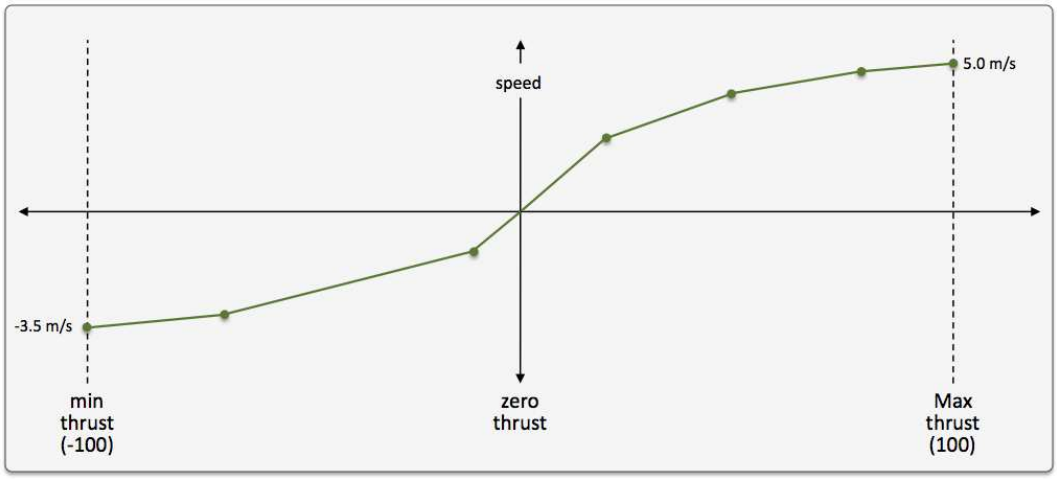


Figure 28: **A Thrust Map**: The example thrust map was defined by seven mapping points in the string "-100:-3.5, -75:-3.2, -10:-2, 20:2.4, 50:4.2, 80:4.8, 100:5".

Automatic Pruning of Invalid Configuration Pairs

The thrust map has an immutable domain of $[-100, 100]$, indicating 100% forward and reverse thrust. Mapping pairs given outside this domain will simply be ignored. The thrust mapping must also be monotonically increasing. This follows the intuition that more positive thrust will not result in the vehicle going slower, and likewise for negative thrust. Since the map is configured with a sequence of pairs as above, a pair that would result in a non-monotonic map is discarded. All maps are created as if they had the pair `0:0` given explicitly. Any pair provided in configuration with zero as the thrust value will ignored; zero thrust always means zero speed. Therefore, the following map configurations would all be equivalent to the map configuration above and shown in Figure 28:

```
THRUST_MAP = -120:-5, -100:-3.5, -75:-3.2, -10:-2, 20:2.4, 50:4.2, 80:4.8, 100:5.0, 120:6
THRUST_MAP = -100:-3.5, -75:-3.2, -10:-2, 20:2.4, 50:4.2, 80:4.8, 90:4, 100:5.0
THRUST_MAP = -100:-3.5, -75:-3.2, -10:-2, 0:0, 20:2.4, 50:4.2, 80:4.8, 100:5.0
THRUST_MAP = -100:-3.5, -75:-3.2, -10:-2, 0:1, 20:2.4, 50:4.2, 80:4.8, 100:5.0
```

In the first case, the pairs `"-120:-5"` and `"120:6"` would be ignored since they are outside the $[-100, 100]$ domain. In the second case, the pair `"90:4"` would be ignored since its inclusion would

entail a non-monotonic mapping given the previous pair of "80:4.8". In the third case, the pair "0:0" would be effectively ignored since it is implied in all map configurations anyway. In the fourth case, the pair "0:1" would be ignored since a mapping from a non-zero speed to zero thrust is not permitted.

Automatic Inclusion of Implied Configuration Pairs

Since the domain $[-100, 100]$ is immutable, the thrust map is altered a bit automatically when or if the user provides a configuration without explicit mappings for the thrust values of -100 or 100. In this case, the missing mapping becomes an implied mapping. The mapping 100:v is added where v is the speed value of the closest point. For example, the following two configurations are equivalent:

```
THRUST_MAP = -75:-3.2, -10:-2, 20:2.4, 50:4.2, 80:4.8
THRUST_MAP = -100:-3.2, -75:-3.2, -10:-2, 20:2.4, 50:4.2, 80:4.8, 100:4.8
```

A Shortcut for Specifying the Negative Thrust Mapping

For convenience, the mapping of positive thrust values to speed values can be used in reverse for negative thrust values. This is done by configuring uSimMarine with THRUST_REFLECT=true, which is false by default. If THRUST_REFLECT is false, then a speed of zero is mapped to all negative thrust values. If THRUST_REFLECT is true, but the user nevertheless provides a mapping for a negative thrust in a thrust map, then the THRUST_REFLECT directive is simply ignored and the thrust map is used instead. For example, the following two configurations are equivalent:

```
THRUST_MAP = -100:-5, -80:-4.8, -50:-4.2, -20:-2.4, 20:2.4, 50:4.2, 80:4.8, 100:5
```

and

```
THRUST_MAP = 20:2.4, 50:4.2, 80:4.8, 100:5
THRUST_REFLECT = true
```

The Inverse Mapping - From Speed To Thrust

Since a thrust map only permits configurations resulting in a non-monotonic function, the inverse also holds (almost) as a valid mapping from speed to thrust. We say "almost" because there is ambiguity in cases where there is one or more plateau in the thrust map as in:

```
THRUST_MAP = -75:-3.2, -10:-2, 20:2.4, 50:4.2, 80:4.8
```

In this case a speed of 4.8 maps to any thrust in the range $[80, 100]$. To remove such ambiguity, the thrust map, as implemented in a C++ class with methods, returns the lowest magnitude thrust in such cases. A speed of 4.8 (or 5 for that matter), would return a thrust value of 80. A speed of -3.2 would return a thrust value of -75. The motivation for this way of disambiguation is that if a thrust value of 80 and 100, both result in the same speed, one would always choose the setting that conserves less energy. Reverse mappings are not used by the uSimMarine application, but may be of use in applications responsible for posting a desired thrust given a desired speed, as with the pMarinePID application.

Default Behavior of an Empty or Unspecified ThrustMap

If `uSimMarine` is configured without an explicit `THRUST_MAP` or `THRUST_REFLECT` configuration, the default behavior is governed as if the following two lines were actually included in the `uSimMarine` configuration block:

```
THRUST_MAP = 100:5
THRUST_REFLECT = false
```

The default thrust map is rendered in Figure 29.

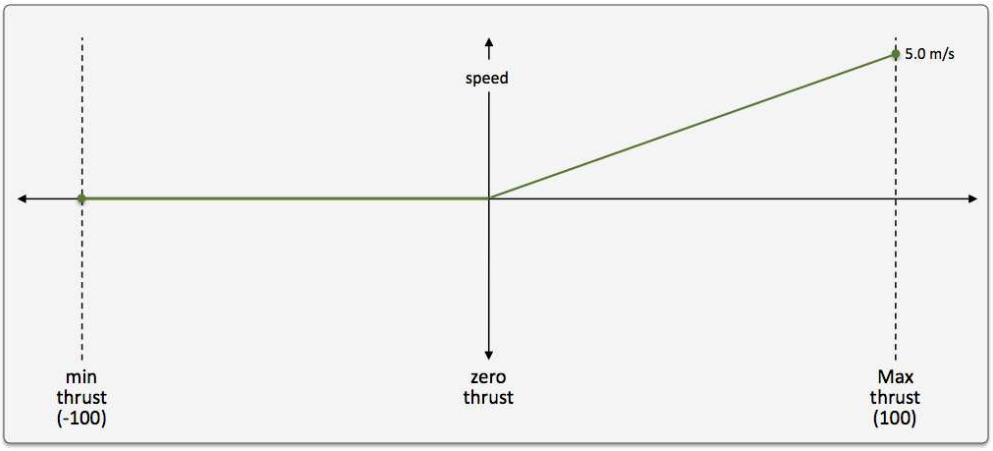


Figure 29: **The Default Thrust Map**: This thrust map is used if no explicit configuration is provided.

This default configuration was chosen for its reasonableness, and to be consistent with the behavior of prior versions of `uSimMarine` where the user did not have the ability to configure a thrust map.

13 The uSimBeaconRange Utility: Simulating Vehicle to Beacon Ranges

The `uSimBne` application is a tool for simulating an on-board sensor that provides a range measurement to a beacon where either (a) theiknows where it is but is trying to determine the position of the beacon via a series of range measurements, or (b) the vehicle dooow where it is but is trying to determine its own position based on the range measurements from one or more beacons at known locat.pskiprange-only sensor may be one that responds to a query, e.g., an acoustic ping, with an immediate reply, e.g. another acoustic ping or echo, which thnge from the source to the beacon is determined by the time-of-flight of the message through the medium, e.g., the approxe speed ound through water. This idea is shown below on the left. Alternatively, if the beacon emits its message on a precise scheduwith a clprecisely synchronized with the vehicle clock, the range measurement may be derived without requiring a separate query from the vehicle. This is the idea behind long baseline acoustic navigation, [3–6]. This idea is shown below on the right.

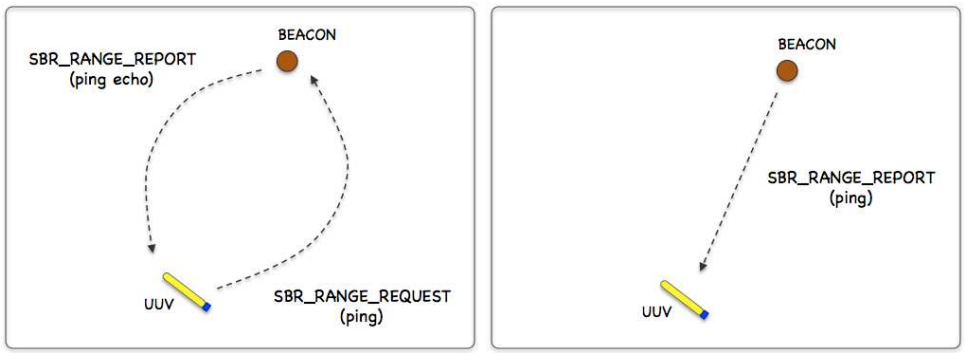


Figure 30: **Beacon Range Sensors:** A vehicle determines its range to a beacon by either (a) emitting a query and waiting for a reply, or (b) waiting for a message to be emitted on fixed schedule. In each case, the time-of-flight of the message through the medium is used to calculate the range.

In the `uSimBeaconRange` application, the beacon and vehicle locations are known to the simulator, and a tidy `SBR_RANGE_REPORT` message is sent to the vehicle(s) as a proxy to the actual range sensor and calculations that would otherwise reside on the vehicle. The MOOS app may be configured to have beacons provide a range report either (a) solicited with a range request, or (b) unsolicited. One may also configure the range at which a range request will be heard, and the range at which a range report will be heard. The app may be further configured to either (1) include the beacon location and ID, or (2) not include the beacon location or ID.

Typical Simulator Topology

The typical module topology is shown in Figure 31 below. Multiple vehicles may be deployed in the field, each periodically communicating with a shoreside MOOS community running a single instance of `uSimBeaconRange`. Each vehicle regularly sends a node report noted by the simulator to keep an updated calculation of each vehicle to each simulated beacon. When a beacon wants to simulate a ping, or range request, it generates the `SBR_RANGE_REQUEST` message send to the shore.

After the simulator calculates the range, a reply message, SBR_RANGE_REPORT is sent to the vehicle.

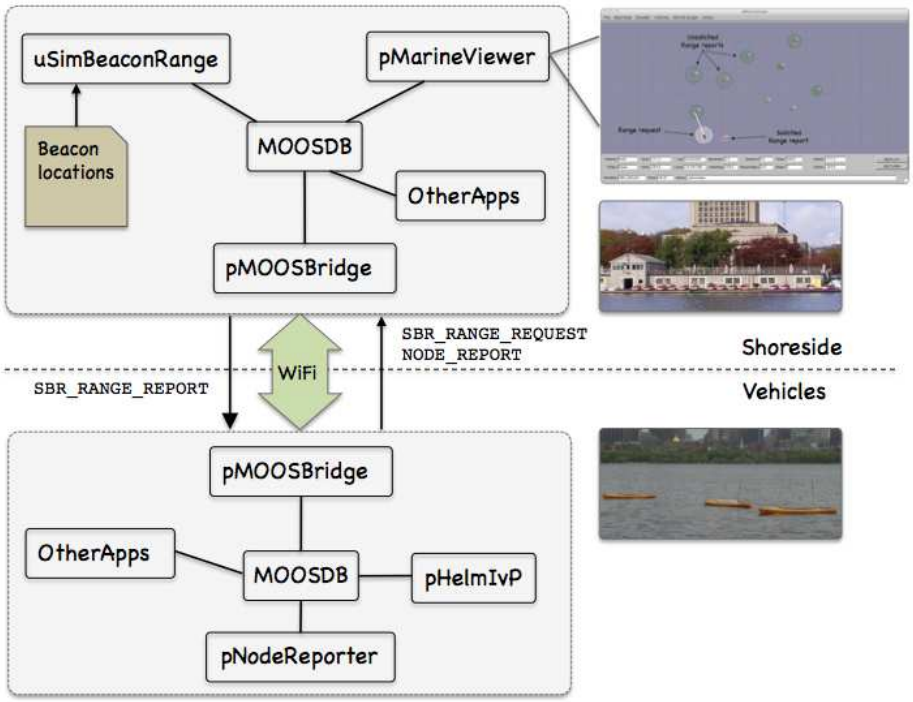


Figure 31: **Typical uSimBeaconRange Topology:** The simulator runs in a shoreside computer MOOS communicty and is configured with the beacon locations. Vehicles accessing the simulator periodically send node reports to the shoreside community. The simulator maintains a running estimate of the range between vehicles and beacons, modulo latency. A vehicle simulates a ping by sending a range request to shore and receiving a range report in return from the simulator.

If running a pure simulation (no deployed vehicles), both MOOS communities may simply be running on the same machine configured with distinct ports. The pMOOSBridge application is shown here for communication between MOOS communities, but there are other alternatives for inter-community communication and the operation of uSimBeaconRange is not dependent on the manner of inter-communication communications.

13.1 Overview of the uSimBeaconRange Interface and Configuration Options

The uSimBeaconRange application may be configured with a configuration block within a .moos file. 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.

13.1.1 Configuration Parameters of uSimBeaconRange

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

BEACON:	Description of beacon location and properties.
DEFAULT_BEACON_FREQ:	Frequency of unsolicited beacon broadcasts ("never").
DEFAULT_BEACON_REPORT_RANGE:	Range at which a vehicle will hear a range report (100).
DEFAULT_BEACON_WIDTH:	Width of beacons (meters) when rendered (4).
DEFAULT_BEACON_COLOR:	Color of beacons when rendered ("red").
DEFAULT_BEACON_SHAPE:	Shape of beacons when rendered ("circle").
PING_PAYMENTS:	How pings treated w.r.t. ping wait time ("upon_response").
GROUND_TRUTH:	If `true`, ground truth is also reported when noise is added.
PING_WAIT:	Mandatory number of seconds between successive vehicle pings.
REACH_DISTANCE:	Range at which a vehicle ping will be heard (100).
REPORT_VARS:	Determines variable name(s) used for range report ("short").
RN_ALGORITHM:	Algorithm for adding random noise to range measurements.
VERBOSE:	If true, verbose status message terminal output (false).

13.1.2 MOOS Variables Published by uSimBeaconRange

The primary output of `uSimBeaconRange` to the MOOSDB is posting of range reports, visual cues for the range reports, and visual cues for the beacons themselves.

SBR_RANGE_REPORT:	A report on the range from a particular beacon to a particular vehicle.
SBR_RANGE_REPORT_NAMEJ:	A report on the range from a particular beacon to vehicle `NAMEJ`.
VIEW_MARKER:	A description for visualizing the beacon in the field. (Section 13.3)
VIEW_RANGE_PULSE:	A description for visualizing the beacon range report. (Section 13.3)

The range report format may vary depending on user configuration. Some examples:

```
SBR\_RANGE_REPORT = "name=alpha,range=129.2,time=19473362764.169"
SBR\_RANGE_REPORT = "name=alpha,range=129.2,id=23,x=54,y=90,time=19473362987.428"
SBR\_RANGE_REPORT_ALPHA = "range=129.2,time=19473362999.761"
```

The vehicle name may be embedded in the MOOS variable name to facilitate distribution of report messages to the appropriate vehicle with `pMOOSBridge`.

13.1.3 MOOS Variables Subscribed for by uSimBeaconRange

The `uSimBeaconRange` application will subscribe for the following four MOOS variables:

SBR_RANGE_REQUEST:	A request to generate range reports for all beacons to all vehicles within range of the beacon.
NODE_REPORT:	A report on a vehicle location and status.
NODE_REPORT_LOCAL:	A report on a vehicle location and status.

13.1.4 Command Line Usage of uSimBeaconRange

The `uSimBeaconRange` 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 command line options may be shown by typing `"uSimBeaconRange --help"`:

Listing 40 - Command line usage for the uSimBeaconRange tool.

```
 0  Usage: uSimBeaconRange file.moos [OPTIONS]
 1
 2  Options:
 3    --alias=<ProcessName>
 4        Launch uSimBeaconRange with the given process
 5        name rather than uSimBeaconRange.
 6    --example, -e
 7        Display example MOOS configuration block
 8    --help, -h
 9        Display this help message.
10    --verbose=Boolean (true/false)
11        Display diagnostics messages. Default is true.
12    --version,-v
13        Display the release version of uSimBeaconRange.
```

13.1.5 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:

```
$ uSimBeaconRange -e
```

This will show the output shown in Listing 41 below.

Listing 41 - Example configuration of the uSimBeaconRange application.

```
 0  ============================================================
 1  uSimBeaconRange Example MOOS Configuration
 2  ============================================================
 3  Blue lines:     Default configuration
 4  Magenta lines:  Non-default configuration
 5
 6  {
 7    AppTick   = 4
 8    CommsTick = 4
 9
10    // Configuring aspects of vehicles in the sim
11    reach_distance = default = 200   // or {nolimit}
12    reach_distance = henry = 40      // meters
13    ping_wait      = default = 30    // seconds
14    ping_wait      = henry   = 120
15    ping_payments  = upon_response   // or {upon_receipt, upon_request}
16
17    // Configuring manner of reporting
18    report_vars    = short  // or {long, both}
```

```
19    ground_truth   = true   // or {false}
20    verbose        = true   // or {false}
21
22    // Configuring default beacon properties
23    default_beacon_shape = circle    // or {square, diamond, etc.}
24    default beacon_color = orange    // or {red, green, etc.}
25    default_beacon_width = 4
26    default_beacon_report_range = 100
27    default_beacon_freq  = never     // or [0,inf]
28
29    // Configuring Beacon properties
30    beacon = x=200, y=435, label=01, report_range=45
31    beacon = x=690, y=205, label=02, freq=90
32    beacon = x=350, y=705, label=03, width=8, color=blue
33
34    // Configuring Artificial Noise
35    rn_algorithm  =  uniform,pct=0   // pct may be in [0,1]
36  }
```

13.2 Using and Configuring the uSimBeaconRange Utility

The `uSimBeaconRange` application is configured primarily with a set of beacons, and a policy for generating range reports to one or more simulated vehicles. The reports may be sent to the vehicles upon a query (solicited) or may be sent unsolicited based on a configured broadcast schedule for each beacon. The possible simulator configuration arrangements are explored by first considering a simple case shown in Figure 32 below, representing a vehicle navigating with three beacons.

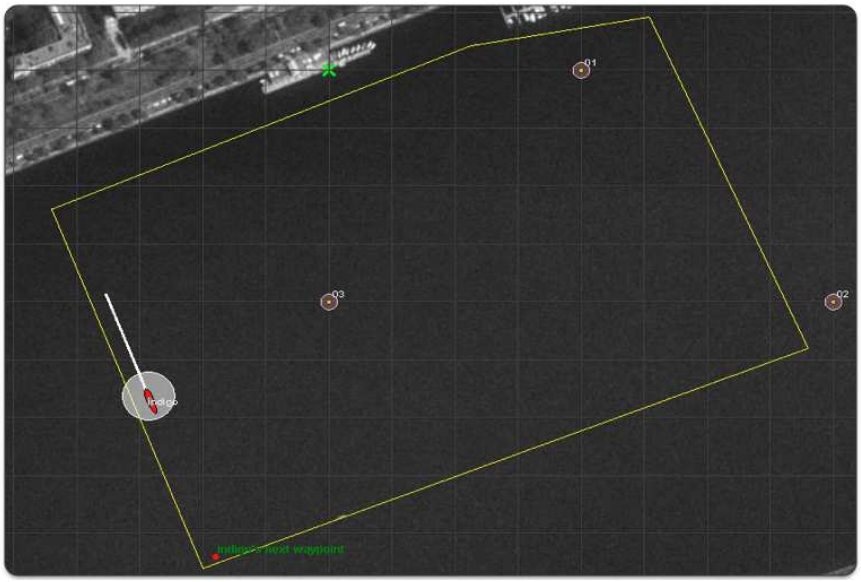


Figure 32: **Simulated LBL Beacons:** Three beacons are simulated, labelled 01, 02, and 03. The vehicle periodically issues a query to which the beacons immediately reply. The `uSimBeaconRange` application handles the queries and generates the range reports sent to each vehicle. The growing circles rendered around the vehicle and beacons represent the generation of the range query and range reports respectively.

The configuration for the uSimBeaconRange is shown in Listing 42 below. The three beacons are configured in lines 19-21. The configuration on line 7 indicates that a beacon query will be heard regardless of the range between the vehicle and the beacon. In the other direction, the range report from the beacon will only be heard if the vehicle is within 100 meters. Line 8 indicates that ping or range request will be honored by the simulator at most once every 30 seconds, and this clock is reset each time a range request is honored by the simulator with the configuration on line 9.

Listing 42 - Example configuration of the uSimBeaconRange application.

```
 1  ProcessConfig = uSimBeaconRange
 2  {
 3    AppTick   = 4   // Standard MOOSApp configurations
 4    CommsTick = 4
 5
 6    // Configuring aspects of the vehicles
 7    reach_distance = default = nolimit
 8    ping_wait      = 30
 9    ping_payments  = upon_accept
10
11    report_vars    = short
12
13    default_beacon_freq  = never       // Only on request (ping)
14    default_beacon_shape = circle
15    default_beacon_color = orange
16    default_beacon_width = 5
17    default_beacon_report_range = 100
18
19    beacon  = label=01, x=200, y=0
20    beacon  = label=02, x=400, y=-200
21    beacon  = label=03, x=0,   y=-200, color=red, shape=triangle, report_range=80
22  }
```

The three beacons in this example are configured on lines 19-21 with unique labels and locations. Each beacon has additional properties, such as its shape, color and width when rendered. Default values for these properties are given in lines 14-16, but may be overridded for a particular beacon as on line 22.

The key configuration line in this example is on line 13 which indicates the beacons by default never generate an unsolicited range report. Reports are only generated upon request. In this example, the simulated vehicle would receive successive groups of SBR_RANGE_REPORT postings from all three simulated beacons, each time the vehicle posts a SBR_RANGE_REQUEST message to the MOOSDB. This example is runnable in the *indigo* example mission distributed with the MOOS-IvP source code and described a bit later in Section 13.4.

13.2.1 Configuring the Beacon Locations and Properties

One or more beacons may be configured by the BEACON configuration parameter provided in the uSimBeaconRange configuration block of the .moos file. Each beacon is configured with a line:

```
BEACON = <configuration>
```

The <configuration> component is a comma-separated list of parameter=value pairs, with the following possible parameters: x, y, label, freq, shape, width, color, and query_range. The following are typical examples:

```
beacon  = x=200, y=260, label=03, freq=10
beacon  = x=-40, y=150, label=04, freq=5:15, color=red, shape=circle, width=4, report_range=200
```

The x and y parameters specify the beacon locations in local coordinates. Like several other MOOS applications, the uSimBeaconRange app looks for a global parameter in the .moos configuration file naming the position of the datum, or 0,0 position in latitude, longitude coordinates. The label parameter provides a unique identifier for the beacon. If a beacon entry is provided using a previously used label, the new beacon will overwrite the prior beacon in the simulator. If no label is provided, an automatic label will be generated equivalent to the index of the new beacon. The freq parameter specifies, in seconds, how often *unsolicited* range reports are generated for each beacon. A simple numerical value may be given, or a colon-separated pair of values as shown above may be used to specify a uniformly random interval of possible durations. The duration between posts will be reset after each post. The report_range parameter specifies the distance, in meters, that a vehicle must be to hear a range report generated by a beacon. The shape parameter indicates the shape used by applications like pMarineViewer when rendering the beacon. The uSimBeaconRange application generates a VIEW_MARKER post to the MOOSDB for each beacon, once upon startup of the simulator. The VIEW_MARKER structure and possible shapes are described in Section 3.4.2. The color parameter specifies the color to be used when rendering the beacon. Legal color strings are described in Appendix B. The width parameter is used to indicate the width, in meters, when rendering the beacon.

For convenience, default values for several of the above properties may be provided with the following five supported configuration parameters:

```
default_beacon_report_range = 100
default_beacon_shape  = circle
default_beacon_color  = orange
default_beacon_width  = 4
default_beacon_freq   = never
```

The above configuration values also represent all the default values. A beacon configuration that includes any of the above five parameters explicitly, will override any default values.

13.2.2 Unsolicited Beacon Range Reports

The simulator may be configured to have all its beacons periodically generate a range report to all vehicles within range. The schedule of reporting may be uniform across all beacons, or individually set for each beacon. The interval of time between reports may also be set to vary according to a uniformly random time interval. By default, beacons are configured to never generate unsolicited range reports unless their frequency parameter is set to something else besides the default value of "never". The default value for all beacons may be configured with the following parameter in the uSimBeaconRange configuration block with the following:

```
default_beacon_freq = 120
```

The parameter value is given in seconds. To configure an interval to vary randomly on each post within a given range, e.g., somewhere between one and two minutes, the following may be used instead:

```
default_beacon_freq = 60:120
```

To configure the simulator to never generate an unsolicited range report, i.e., only solicited reports, use the following:

```
default_beacon_freq = never
```

Upon each range report post to the MOOSDB, the interval until the next post is recalculated. The beacon schedule may also be configured to be unique to a given beacon. The beacon configuration line accepts the `freq` parameter as described earlier in Section 13.2.1. The configuration provided for an individual beacon overrides the default frequency configuration.

Once a beacon has generated a report, it will not generate another *unsolicited* report until after the prevailing time interval has passed. However, if the simulator detects that the beacon has been solicited for a range report via an explicit range request from a nearby vehicle, a range report may be generated immediately. In this case the clock counting down to the beacon's next unsolicited report is reset.

13.2.3 Solicited Beacon Range Reports

The `uSimBeaconRange` application accepts requests from vehicles, and may or may not generate one or more range reports for beacons within range of the vehicle making the request. In short, things operate like this: (a) a range request is received by `uSimBeaconRange` through its mailbox on the variable `RANGE_REQUEST`, (b) a determination is made as to whether the request is within range of the beacon and whether the request is allowed based on limits on the frequency of range requests, (c) a range report is generated and posted to the variable `SBR_RANGE_REPORT`. The following is an example of the range request format:

```
SBR_RANGE_REQUEST = "name=charlie"
```

Note that if a vehicle generates a range request triggering a range report from a beacon, the range report is sent to *all* vehicles within range of the beacon. Presumably the simulator has also received, at some point in the past, a node report, typically generated from the `pNodeReporter` application (Section 10) running on the vehicle. So the simulator not only knows which vehicle is making the range request, but also where that vehicle is located. It needs the vehicle location to determine the range between the vehicle and beacon, to generate the requested range report. The simulator also uses this range information to decide if it wants regard the beacon as being close enough to hear the request, and whether the vehicle is close enough to the beacon to hear the report.

13.2.4 Limiting the Frequency of Vehicle Range Requests

From the perspective of operating a vehicle, one may ask: why not request a range report from all beacons as often as possible? There may be reasons why this is not feasible outside simulation.

Limits may exist due to power budgets of the vehicle and/or beacons, and there may be prevailing communications protocols that make it at least impolite to be pushing range requests through a shared communications medium.

To reflect this limitation, the uSimBeaconRange application may be configured to limit the frequency in which a vehicle's range request (or ping) will be honored with a range report reply. By default this frequency is set to once every 30 seconds for all vehicles. The default for all vehicles may be changed with the following configuration in the .moos file:

```
PING_WAIT = default = 60
```

If the time interval as above is set to 60 seconds, what happens if a vehicle requests a range report 40 seconds after its previous request? Is it simply ignored, needing to wait another 20 seconds? Or is the clock reset to zero forcing the vehicle to wait 60 seconds before a ping is honored? By default, the former is the case, but the simulator may be configured to the more draconian option with:

```
PING_PAYMENTS = upon_request
```

Suppose the minimum time interval has elapsed, but the querying/pinging vehicle is too far out of range from any beacon to hear even a single range report. Will the result be that the clock is reset to zero, forcing the vehicle to wait another 60 seconds before a query is honored? By default this is the case, but the simulator may be configured to not reset the clock unless the querying vehicle has received at least one range report for its query:

```
PING_PAYMENTS = upon_response
```

In short, the simulator configuration parameter, PING_PAYMENTS, may be configured with one of three options, "upon_request", "upon_response", or "upon_accept", with the default being the latter.

13.2.5 Producing Range Measurements with Noise

In the default configuration of uSimBeaconRange, range reports are generated with the most precise range estimate as possible, with the only error being due to the latency of the communications generating the range request and range report. Additional noise/error may be added in the simulator for each range report with the following configuration parameter:

```
rn_algorithm   = uniform,pct=0.12     // Values in the range [0,1]
```

Currently the only noise algorithm supported is the generation of uniformly random noise on the range measurement. The noise level, θ, set with the parameter rn_uniform_pct, will generate a noisy range from an otherwise exact range measurement r, by choosing a value in the range $[\theta r, r + \theta r]$. The range without noise, i.e., the ground truth, may also be reported by the simulator if desired by setting the configuration parameter:

```
ground_truth = true
```

This will result in an additional MOOS variables posted, SBR_RANGE_REPORT_GT, with the same format as SBR_RANGE_REPORT, except the reported range will be given without noise.

13.2.6 Console Output Generated by uSimBeaconRange

Information regarding the startup of the `uSimBeaconRange` application may be monitored from an open console window where `uSimBeaconRange` is launched. If the verbose setting is turned on, further output is generated as the simulator progresses and receives range requests and generates range reports. The verbose setting may be turned on from the command line, `--verbose`, or in the mission file configuration block with `verbose=true`. Example output is shown below in Listing 43. Certain startup information common across all MOOS applications can be found in lines 1-14, and lines 33-36 in the example below. The block of output in lines 16-31 provides startup feedback unique to `uSimBeaconRange`. This cannot be turned off with the verbose setting, and should appear in blue in the console window. The Figlog summary in lines 17-22 provides feedback on the the configuration parameters provided in the `uSimBeaconRange` block of the mission file. Following this, e.g. in lines 23-30, a summary of the simulator model is given, showing the configured beacons, rendering hints, and other simulator policies.

Listing 43 - Example uSimBeaconRange console output.

```
 1  ************************************************
 2  *                                              *
 3  *       This is MOOS Client                    *
 4  *       c. P Newman 2001                       *
 5  *                                              *
 6  ************************************************
 7
 8  ---------------MOOS CONNECT----------------------
 9    contacting a MOOS server localhost:9000 -  try 00001
10    Contact Made
11    Handshaking as "uSimBeaconRange"
12    Handshaking Complete
13    Invoking User OnConnect() callback...ok
14  ------------------------------------------------
15
16  Simulated Range Sensor starting...
17  ================================================
18  Figlog Summary:
19  Messages: (0)
20  Warnings: (0)
21  Errors: (0)
22  ================================================
23  SRS Model - # Beacons: 3
24    [0]:x=0,y=-200,label=03,color=orange,type=circle,width=4
25    [1]:x=400,y=-200,label=02,color=orange,type=circle,width=4
26    [2]:x=200,y=0,label=01,color=orange,type=circle,width=4
27  Default Beacon Color: orange
28  Default Beacon Shape: circle
29  Default Beacon Width: 4
30   NodeRecords:: 0
31  Simulated Range Sensor started.
32
33  uSimBeaconRange is Running:
34          AppTick   @ 4.0 Hz
35          CommsTick @ 4 Hz
36          Time Warp @ 6.0
```

```
37
38 Received first NODE_REPORT for: indigo
39 ***************************
40 Range request received from: indigo
41    Elapsed time: 167.24879
42    Query accepted by uSimBeaconRange.
43    Range report sent from beacon[03] to vehicle[indigo]
44    Range report sent from beacon[02] to vehicle[indigo]
45    Range report sent from beacon[01] to vehicle[indigo]
46 ***********************
47 Unsolicited beacon reports:
48    Range report sent from beacon[03] to vehicle[indigo]
49    Range report sent from beacon[01] to vehicle[indigo]
50 ****************************************
```

Unless the verbose setting is turned on, the output ending on line 37 above should be the last output written to the console for the duration of the simuator.

In the verbose mode, the simulator will produce event-based output as shown in the example above beginning on line 38. The asterisks in lines 39, 46, and 50 are not merely visual separators. An asterisk represents a single receipt of a `NODE_REPORT` message. Receiving node reports is essential for the operation of the simulator and this provides a bit of visual verification that this is occurring. Presumably node reports are being received much more often than range requests and range reports are handled, as is the case in the above example. The first time a node report is received for a particular vehicle, an announcement is made as shown on line 38.

In addition to handling incoming node reports, on any given iteration, the simulator may also handle an incoming range request, or may generate an unsolicited range report based on a beacon schedule. Console output for incoming range requests may look like that shown in lines 40-45 above. First the range request and the requesting vehicle is announced as online 40. The elapsed time since the vehicle last made a range request is shown as on line 41. If the request is honored by the simulator, this is indicated as shown on line 42. Otherwise a reason for denial may be shown. If the query is accepted, range reports may be generated for one or more vehicles. For each such vehicle, a line announcing the new report is generated, as in lines 43-45. On an iteration where unsolicited range reports are generated, output similar to that shown in lines 47-49 will generated. For each report, the beacon and receiving vehicle are named.

13.3 Interaction between uSimBeaconRange and pMarineViewer

The `uSimBeaconRange` application will post certain messages to the MOOSDB that may be subscribed for by GUI based applications like `pMarineViewer` for visualizing the posting of `SBR_RANGE_REPORT` and `SBR_RANGE_REQUEST` messages, as well as visualizing the beacon locations. A snapshot of the `pMarineViewer` window is shown below, with one vehicle and several beacons.

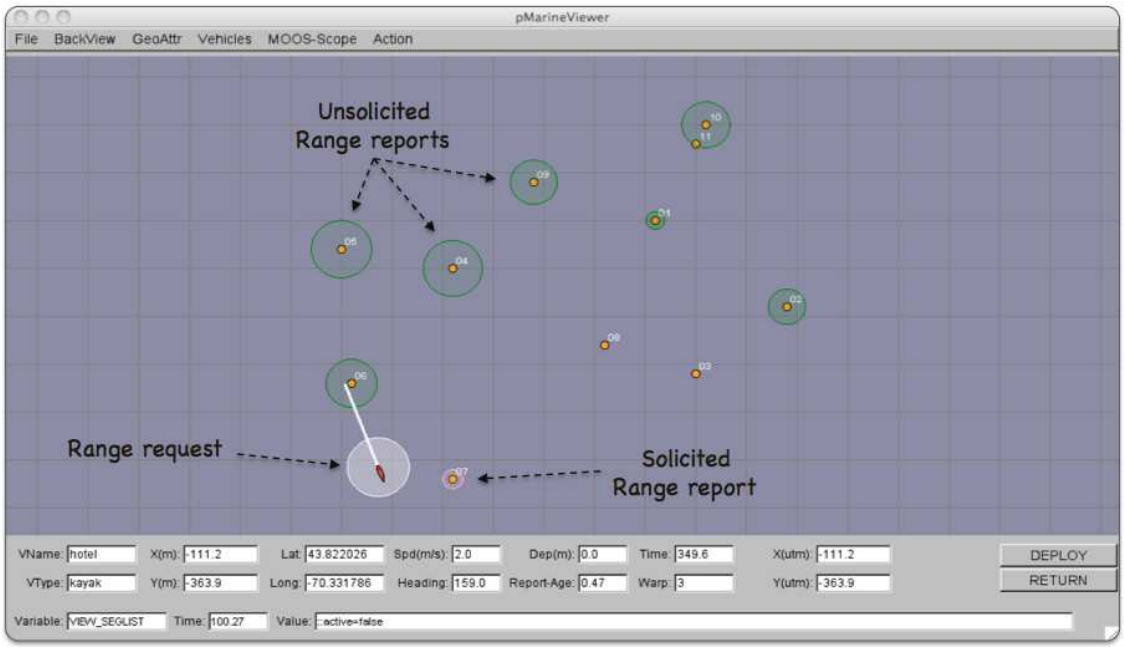


Figure 33: **Beacons in the pMarineViewer:** The `VIEW_RANGE_PULSE` message is passed to pMarineViewer to render unsolicited range reports (here in green), range requests from a vehicle (here in white), and solicited range reports in response to a range requeste (here in pink). The viewer alse renders the beacons and their labels upon receiving `VIEW_MARKER` messages posted by the uSimBeaconRange application. The pulses are only momemtarily visible until another `VIEW_RANGE_PULSE` message is received.

The `VIEW_MARKER` Data Structure

The `uSimBeaconRange` application, upon startup, posts the beacon locations in the form of the `VIEW_MARKER` data structure. This MOOS variable is one of the default variables registered for by the `pMarineViewer` application. The types of supported markers are described in Section 3.4.2. The marker type, size and color are configurable in the `uSimBeaconRange` configuration block. The user may use the variation in marker rendering to correspond to variation in beacon reporting or querying characteristics.

The `RANGE_PULSE` Data Structure

A *range pulse* message is used by the `uSimBeaconRange` application to convey visually the generation of a range report, or the receipt of a range request. The pulse is rendered as a ring with a growing radius having either the beacon or the vehicle at the center. A pulse eminating from a beacon

indicates a range report, and a pulse eminating from a vehicle indicates a range request. By default different colors may be used for solicited and unsolicited range reports. In Figure 33 for example, the green rings represent unsolicited reports, the white ring represents a range request made by the vehicle, and the pink ring represents a response to the range request made by the one beacon within range to the vehicle.

The `RANGE_PULSE` message is a data structure implemented in the `XYRangePulse` class, and usually passed through the MOOS variable `VIEW_RANGE_PULSE` with the following format:

```
VIEW_RANGE_PULSE = <pulse>
```

The `<pulse>` component is a series of comma-separated `parameter=value` pairs. The supported parameters are: `x`, `y`, `radius`, `duration`, `time`, `fill`, `fill_color`, `label`, `edge_color`, and `edge_size`.

The `x` and `y` parameters convey the center of the pulse. The `radius` parameter indicates the radius of the circle at its maximum. The `duration` parameter is the number of seconds the pulse will be rendered. The pulse will grow its radius linearly from zero meters at zero seconds to `radius` meters at `duration` seconds. The `fill` parameter is in the range $[0, 1]$, where 0 is full transparency (clear) and 1 is fully opaque. The pulse transparency increases linearly as the range ring is rendered. The starting transparency at $radius = 0$ is given by the `fill` parameter. The transparency at the maximum radius is zero. The `fill_color` parameter specifies the color rendered to the internal part of the range pulse. The choice of legal colors is described in Appendix B. The `label` is a string that uniquely identifies the range instance to consumers like `pMarineViewer`. As with other objects in `pMarineViewer`, if it receives an object the same label and type as one previously received, it will replace the old object with the new one in its memory. The `edge_color` parameter describes the color of the ring rendered in the range pulse. The `edge_size` likewise describes the line width of the rendered ring. The `time` parameter indicates the UTC time at which the range pulse object was generated.

Below is an example string representing a range pulse. Each field, with the exception of the x, y position, also indicates the default values if left unspecified:

```
VIEW_RANGE_PULSE = x=-40,y=-150,radius=40,duration=15,fill=0.25,fill_color=green,label=04
                   edge_color=green,time=3892830128.5,edge_size=1
```

Exercise 13.1: Poking a RangePulse for visualizing in pMarineViewer.

- Try running the Alpha mission described in [2]. The `uPokeDB` tool is described in Section 8.
- Poke the MOOSDB with:

  ```
  $ uPokeDB alpha.moos VIEW_RANGE_PULSE="x=100,y=-50,radius=40,duration=15,fill=0.25,
  fill_color=green,label=04,time=@MOOSTIME"
  ```

  A range pulse should appear in the viewer 100 meters East and 50 meters South of the vechicle's starting position. Note the special macro `@MOOSTIME`, which uPokeDB will expand to the UTC time at which the poke was made.

13.4 The Indigo Example Mission Using uSimBeaconRange

The *indigo* mission is distributed with the MOOS-IvP source code and contains a ready example of the `uSimBeaconRange` application, configured with three beacons acting as long baseline (LBL) beacons as described at the beginning of Section 13.2. Assuming the reader has downloaded the source code available at `www.moos-ivp.org` and built the code according to the discussion in Section 1.4, the *indigo* mission may be launched by:

```
$ cd moos-ivp/ivp/missions/s9_indigo/
$ ./launch.sh 10
```

The argument, 10, in the line above will launch the simulation in `10x` real time. Once this launches, the `pMarineViewer` GUI application should launch and the mission may be initiated by hitting the `DEPLOY` button. The vehicle will traverse a survey pattern over the rectangular operation region shown in Figure 32, periodically generating a range request to the three beacons. With each range request, a white range pulse should be visible around the vehicle. Almost immediately afterwards, a range report for each beacon is generated and a red range pulse around each beacon is rendered. The snapshot in Figure 32 depicts a moment in time where the range report visual pulses are beginning to grow around the beacons and the range request visual pulse is still visible around the one vehicle.

How does the simulated vehicle generate a range request? In practice a user may implement an intelligent module to reason about when to generate requests, but in this case the `uTimerScript` application is used by creating a script that repeats endlessly, generating a range request once every 25-35 seconds. The script is also conditioned on `(NAV_SPEED > 0)`, so the pinging doesn't start until the vehicle is deployed. The configuration for the script can be seen in the `uTimerScript` configuration block in the `indigo.moos` file. More on the `uTimerScript` application can be found in Section 9.

Examining the Log Data from the Indigo Mission

After the launch script above has launched the simulation, the script should leave the console user with the option to "Exit and Kill Simulation" by hitting the '2' key. Once the vehicle has been deployed and traversed to one's satisfaction, exit the script. A log directory should have been created by the `pLogger` application in the directory where the simulation was launched. The directory name should be begin with `MOOSLog_` with the remainder of the directory name composed from the timestamp of the launch.

Let's take a look at some of the data related to the simulation and `uSimBeaconRange` in particular. A dump of the entire file reveals a deluge of information. To look at the information relevant to the `uSimBeaconRange` application, the file is pruned with the `aloggrep` tool described in Section 16.5:

```
$ aloggrep MOOSLog_21_2_2011_____22_32_48.alog uSimBeaconRange uTimerScript
```

This produces a subset of the alog file similar to that shown in Listing 44, showing only log entries made by either the `uSimBeaconRange` application, or the `uTimerScript` application which generated all the range requests as described above. The first three posts made to the MOOSDB

by uSimBeaconRange are the VIEW_MARKER posts representing a visual cue for the pMarineViewer application to render the three beacons.

Listing 44 - A subset of the data logged from the Indigo example mission's alog file.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% LOG FILE:       ./MOOSLog_22_2_2011_____17_27_06/MOOSLog_22_2_2011_____17_27_06.alog
%% FILE OPENED ON  Tue Feb 22 17:27:06 2011
%% LOGSTART               23371445284.8
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
55.697       VIEW_MARKER          uSimBeaconRange x=0,y=-200,label=03,color=orange,type=circle,width=4
55.698       VIEW_MARKER          uSimBeaconRange x=400,y=-200,label=02,color=orange,type=circle,width=4
55.698       VIEW_MARKER          uSimBeaconRange x=200,y=0,label=01,color=orange,type=circle,width=4
100.663      SBR_RANGE_REQUEST    uTimerScript    name=indigo
100.846      VIEW_RANGE_PULSE     uSimBeaconRange x=-97.65,y=-64.84,radius=50,duration=6,...
100.846      SBR_RANGE_REPORT     uSimBeaconRange vname=indigo,range=166.7446,id=03,time=23371445385.636
100.846      VIEW_RANGE_PULSE     uSimBeaconRange x=0,y=-200,radius=40,duration=15,...
100.846      SBR_RANGE_REPORT     uSimBeaconRange vname=indigo,range=515.6779,id=02,time=23371445385.636
100.846      VIEW_RANGE_PULSE     uSimBeaconRange x=400,y=-200,radius=40,duration=15,...
100.847      SBR_RANGE_REPORT     uSimBeaconRange vname=indigo,range=304.6305,id=01,time=23371445385.636
100.847      VIEW_RANGE_PULSE     uSimBeaconRange x=200,y=0,radius=40,duration=15,...
160.419      SBR_RANGE_REQUEST    uTimerScript    name=indigo
160.597      VIEW_RANGE_PULSE     uSimBeaconRange x=-197.96,y=-129.16,radius=50,duration=6,...
160.597      SBR_RANGE_REPORT     uSimBeaconRange vname=indigo,range=210.2533,id=03,time=23371445445.392
160.597      VIEW_RANGE_PULSE     uSimBeaconRange x=0,y=-200,radius=40,duration=15,...
160.597      SBR_RANGE_REPORT     uSimBeaconRange vname=indigo,range=602.1416,id=02,time=23371445445.392
160.597      VIEW_RANGE_PULSE     uSimBeaconRange x=400,y=-200,radius=40,duration=15,...
160.597      SBR_RANGE_REPORT     uSimBeaconRange vname=indigo,range=418.3951,id=01,time=23371445445.392
160.597      VIEW_RANGE_PULSE     uSimBeaconRange x=200,y=0,radius=40,duration=15,...
```

The first range request is generated at time 100.663 by the uTimerScript. The uSimBeaconRange application receives this mail and posts a range pulse at time 100.846 conveying the range request from the vehicle, e.g., the white circle in Figure 32. The range request is met immediately and two posts are generated for each beacon. The VIEW_RANGE_PULSE message indicates the simulator has generated a range report for the beacon (the red circle in Figure 32. The SBR_RANGE_REPORT message is the actual range report to be used by the vehicle as it sees fit.

13.4.1 Generating Range Report Data for Matlab

The log files generated as in Listing 44 above may processed to form a table of values suitable for Matlab processing. The alog2rng tool may be run on an alog file from the command line:

```
$ alog2rng MOOSLog_21_2_2011_____22_32_48.alog
```

This will generate a table of data like that below. The left column is the timestamp from the log file. The next N columns are the range measurements from each beacon. And the last three columns are the "ground truth" vehicle position and heading. The last three columns may be excluded with the --nav=false switch on the command line.

```
Time       03         02         01         NAV_X      NAV_Y      NAV_HDG
100.846    166.7446   515.6779   304.6305   -99.371    -65.917    238.000
160.597    210.2533   602.1416   418.3951   -198.538   -130.397   197.456
224.844    163.4205   558.5143   433.6937   -155.744   -248.991   159.000
```

The alog2rng tool is part of the Alog-Toolbox described in Section 16 along with other tools for examining and modifying alog files generated by pLogger.

14 The uSimContactRange Utility: Detecting Contact Ranges

The `uSimContactRange` application is a tool for simulating range measurements to off-board contacts, as a proxy for an on-board active sonar sensor. The range-only measurements are provided conditionally, depending on the range between the pinging vehicle and the contact. The simulator may optionally be configured to provide range measurements with noise.

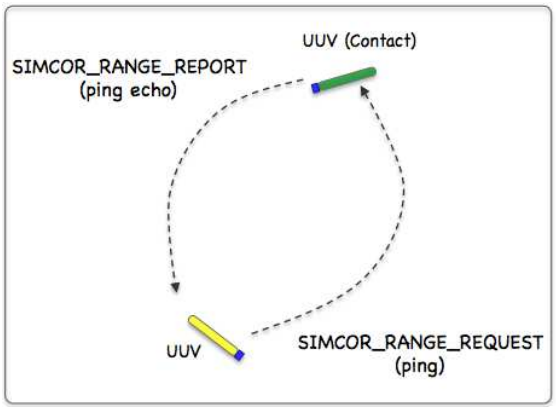


Figure 34: **Simulated Active Sonar:** A vehicle determines its range to another vehicle by producing a simulated sonar ping (a range request to the simulator), and the simulator conditionally responds to the querying vehicle with a report containing the range to nearby vehicles. All vehicles send frequent and regular node reports to the simulator so the simulator can report the range between any two vehicles at any time. The simulator may or may not reply to the range request depending on the range between the two vehicles and thresholds configured by the user.

In the `uSimContactRange` application, the beacon and vehicle locations are known to the simulator, and a tidy `RANGE_REPORT` message is sent to the vehicle(s) as a proxy to the actual range sensor and calculations that would otherwise reside on the vehicle. The MOOS app may be configured to have beacons provide a range report either (a) solicited with a range request, or (b) unsolicited. One may also configure the range at which a range request will be heard, and the range at which a range report will be heard. The app may be further configured to either (1) include the beacon location and ID, or (2) not include the beacon location or ID.

Typical Simulator Topology

The typical module topology is shown in Figure 35 below. Multiple vehicles may be deployed in the field, each periodically communicating with a shoreside MOOS community running a single instance of `uSimContactRange`. Each vehicle regularly sends a node report noted by the simulator to keep an updated calculation of each vehicle to each simulated beacon. When a vehicle wants to simulate a ping, or range request, it generates the `SIMCOR_RANGE_REQUEST` message sent to the shore. After the simulator calculates the range, a reply message, `SIMCOR_RANGE_REPORT` message is sent to the vehicle, using `pMOOSBridge` or similar app.

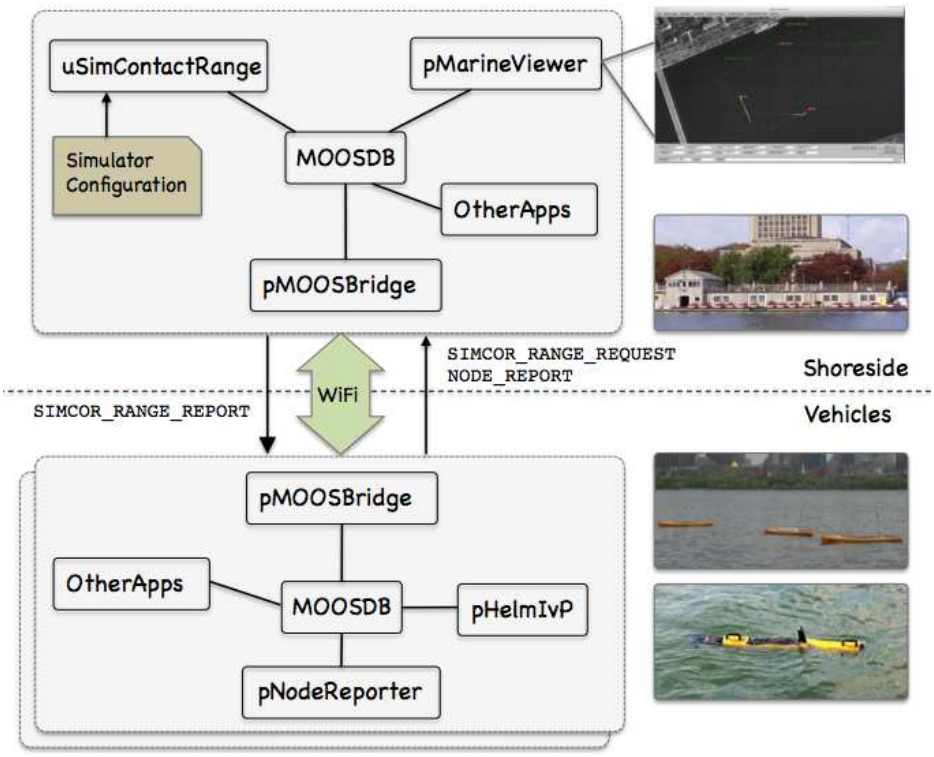


Figure 35: **Typical uSimContactRange Topology:** The simulator runs in a shoreside computer MOOS community. All deployed vehicles periodically send node reports to the shoreside community. The simulator maintains a running estimate of the range between vehicles, modulo latency. A vehicle simulates a ping by sending a range request to shore and receiving a range report in return from the simulator. The simulator also posts visual artifacts (`VIEW_RANGE_PULSE` messages) read by the pMarineViewer app optionally running shoreside.

If running a pure simulation (no physically deployed vehicles), both MOOS communities may simply be running on the same machine configured with distinct ports. The `pMOOSBridge` application is shown here for communication between MOOS communities, but there are other alternatives for inter-community communication and the operation of `uSimContactRange` is not dependent on the manner of inter-communication communications.

14.1 Overview of the uSimContactRange Interface and Configuration Options

The `uSimContactRange` application may be configured with a configuration block within a `.moos` file. 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.

14.1.1 Configuration Parameters of uSimContactRange

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

REACH_DISTANCE:	Distance out to which the pinging vehicle will be heard (100).
REPLY_DISTANCE:	Distance out to which the pinged vehicle will be heard (100).
PING_WAIT:	Minimum seconds enforced between pings (30).
PING_COLOR:	Visual preference: color of initiating ping message (white).
REPLY_COLOR:	Visual preference: color of replying message (chartreuse).
RN_ALGORITHM:	Algorithm for adding random noise to the range measurement
REPORT_VARS:	Determines variable name(s) used for range report ("short").
VERBOSE:	If true, verbose status message terminal output (false).

14.1.2 MOOS Variables Published by uSimContactRange

The primary output of uSimContactRange to the MOOSDB is posting of range reports and visual cues for the range reports.

SIMCOR_RANGE_REPORT:	A report on the range from a particular vehicle to the pinging vehicle.
SIMCOR_RANGE_REPORT_NAMEJ:	A report on the range from a particular named NAMEJ, to the pinging vehicle.
VIEW_RANGE_PULSE:	A description for visualizing the beacon range report. (Section 13.3)

The range report format may vary depending on user configuration. Some examples:

```
SIMCOR_RANGE_REPORT = "name=alpha,range=129.2,time=19473362764.169"
SIMCOR_RANGE_REPORT = "name=alpha,range=129.2,target=jackal,x=54,y=90,time=19473362987.428"
SIMCOR_RANGE_REPORT_ALPHA = "range=129.2,time=19473362999.761"
```

The name of the vehicle requesting the report (generating the ping) may be embedded in the MOOS variable name to facilitate distribution of report messages to the appropriate vehicle with pMOOSBridge.

14.1.3 MOOS Variables Subscribed for by uSimContactRange

The uSimContactRange application will subscribe for the following four MOOS variables:

SIMCOR_RANGE_REQUEST:	A request to generate range reports for all targets to all vehicles within range of the target.
NODE_REPORT:	A report on a vehicle location and status.
NODE_REPORT_LOCAL:	A report on a vehicle location and status.

14.1.4 Command Line Usage of uSimContactRange

The uSimContactRange 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 command line options may be shown by typing "uSimContactRange --help":

Listing 45 - Command line usage for the uSimContactRange tool.

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

14.2 Configuring the uSimContactRange Parameters

The uSimContactRange simulator has two range configuration parameters:

```
REACH_DISTANCE = default = <meters>    // or "nolimit"
REPLY_DISTANCE = default = <meters>    // or "nolimit"
```

The simulator uses these parameters to decide whether a ping (range request) will reach other nearby vehicles. Two separate range parameters are used so that the simulator can support scenarios where some vehicles have a more powerful sonar than others, and some targets are harder to detect (absorb or deflect more energy) than others. Both parameters are given in meters.

When a range request is received by the simulator, it knows which vehicle is making the request since the requesting vehicle's name comprises the range request. For example:

```
MOOS Variable         Community   Value
-------------         ---------   -----
SIMCOR_RANGE_REQUEST   archie      name=archie
```

The simulator maintains a record of the location of all known vehicles, by receiving NODE_REPORT messages regularly from each vehicle known to the simulator. When a range request is made, the simulator looks up the REACH_DISTANCE for the requesting vehicle, and the REPLY_DISTANCE for the target vehicle and if the sum, REACH_DISTANCE + REPLY_DISTANCE is less than the actual present range between the two vehicles, a range report is generated. For example:

```
MOOS Variable         Community   Value
-------------         ---------   -----
SIMCOR_RANGE_REPORT    shoreside   vname=archie,range=126.54,target=jackal,time=19656022406.44
```

If the user provides no configuration parameters, all vehicles will default to have the same reach and reply distances of 100 meters. The default values may be overridden with something like:

```
REACH_DISTANCE = default = 120
REPLY_DISTANCE = default = 80
```

The above two lines, in effect, are the same as REACH_DISTANCE = 100 and REPLY_DISTANCE = 100. Things become interesting when individual vehicles are given values different from the default. Consider the more advantageously configured vehicle, *victor*, below:

```
REACH_DISTANCE = victor = 250
REPLY_DISTANCE = victor = 20
```

If either the reach or reply distance for a given pair of vehicles is set to `nolimit`, then a range report will always be generated regardless of current range between the two vehicles. Future enhancements to this simulator module may include the factoring of vehicle speed and relative bearing to one another in the threshold determination of sending range reports.

14.3 Limiting the Frequency of Vehicle Range Requests

From the perspective of operating a vehicle, one may ask: why not request a range report (ping) as often as possible? There may be reasons why this is not feasible outside simulation. Limits may exist due to power budgets of the vehicle, and there may be prevailing protocols that make it at least impolite to be frequently pinging.

To reflect this limitation, the `uSimContactRange` utility may be configured to limit the frequency in which a vehicle's range request (or ping) will be honored with a range report reply. By default this frequency is set to once every 30 seconds for all vehicles. The default for all vehicles may be changed with the following configuration in the `.moos` file:

```
PING_WAIT = default = 60
```

The limits for a particular vehicle may be set with a similar configuration line:

```
PING_WAIT = henry = 90
```

14.4 Producing Range Measurements with Noise

In the default configuration of `uSimContactRange`, range reports are generated with the most precise range estimate as possible, with the only error being due to the latency of the communications generating the range request and range report. Additional noise/error may be added in the simulator for each range report with the following configuration parameter:

```
rn_algorithm   = uniform,pct=0.12     // Values in the range [0,1]
```

Currently the only noise algorithm supported is the generation of uniformly random noise on the range measurement. The noise level, θ, set with the parameter `rn_uniform_pct`, will generate a noisy range from an otherwise exact range measurement r, by choosing a value in the range $[\theta r, r + \theta r]$. The range without noise, i.e., the ground truth, may also be reported by the simulator if desired by setting the configuration parameter:

```
ground_truth = true
```

This will result in an additional MOOS variables posted, `SIMCOR_RANGE_REPORT_GT`, with the same format as `SIMCOR_RANGE_REPORT`, except the reported range will be given without noise.

14.5 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:

```
$ uSimContactRange -e
```

This will show the output shown in Listing 46 below.

Listing 46 - Example configuration of the uSimContactRange application.

```
 0  ============================================================
 1  uSimContactRange Example MOOS Configuration
 2  ============================================================
 3  Blue lines:     Default configuration
 4  Magenta lines:  Non-default configuration
 5
 6  ProcessConfig = uSimContactRange
 7  {
 8    AppTick   = 4
 9    CommsTick = 4
10
11    // Configuring aspects of the vehicles in the sim
12    reach_distance = default = 100  // in meters or {nolimit}
13    reply_distance = default = 100  // in meters or {nolimit}
14    ping_wait      = default = 30   // in seconds
15
16    // Configuring manner of reporting
17    report_vars    = short // or {long, both}
18    ground_truth   = true  // or {false}
19    verbose        = true  // or {false}
20
21    // Configuring visual artifacts
22    ping_color     = white
23    reply_color    = chartreuse
24
25    // Configuring Artificial Noise
26    rn_algorithm   = uniform,pct=0
27  }
```

The console output uses color to discern default configurations from non-default configurations. Any line rendered as a default configuration may be omitted without any net change to the configuration.

14.5.1 Console Output Generated by uSimContactRange

Information regarding the startup of the `uSimContactRange` application may be monitored from an open console window where `uSimContactRange` is launched. If the verbose setting is turned on, further output is generated as the simulator progresses and receives range requests and generates range reports. The verbose setting may be turned on from the command line, `--verbose`, or in the mission file configuration block with `verbose=true`. Example output is shown below in Listing 47. Certain startup information, common across all MOOS application,s can be found in lines

0-13. The block of output in lines 16-36 provides startup information specific to `uSimContactRange`. This cannot be turned off with the verbose setting, and should appear in blue in the console window. These lines convey the configuration settings of `uSimContactRange` read from the MOOS configuration block like the one shown previously in Listing 46.

Listing 47 - Example uSimContactRange console output.

```
 0  ************************************************
 1  *                                              *
 2  *       This is MOOS Client                    *
 3  *       c. P Newman 2001                       *
 4  *                                              *
 5  ************************************************
 6
 7  ---------------MOOS CONNECT----------------------
 8    contacting a MOOS server localhost:9200 -  try 00001
 9    Contact Made
10    Handshaking as "uSimContactRange"
11    Handshaking Complete
12    Invoking User OnConnect() callback...ok
13  ------------------------------------------------
14
15
16  Simulated Active Sonar starting...
17  ========================================
18  Acoustic Sonar Simulator Model:
19  ========================================
20  Default Reach Distance:   100
21  Default Reply Distance:   100
22  Default Ping Wait:        30
23  Random Noise Algorithm:   uniform
24  Random Noise Uniform Pct: 0.04
25  Ping Color:               white
26  Reply Color:              chartreuse
27  Ground Truth Reporting:   true
28  ReachMap:
29    VName: archie  Dist: 190
30  ReplyMap:
31    VName: jackal  Dist: 50
32  PingWaitMap:
33    VName: archie  PingWait: 125
34  Contact Records: 0
35  ========================================
36  Simulated Active Sonar started.
37
38  uSimContactRange is Running:
39           AppTick   @ 4.0 Hz
40           CommsTick @ 4 Hz
41  %*%************************************************************
42  ****************************************************************
43  **********************
44  Range request received from: archie
45     Elapsed time: 46.04512
46     Range Request accepted by uSimContactRange.
47     Range Request resets the clock for vehicle.
48     Range Report sent from targ vehicle[jackal] to receiver vehicle[archie]
49  DBPost: VIEW_RANGE_PULSE  x=-40.53,y=-33.66,radius=50,duration=6,fill=,label=archie_ping,
50                            edge_color=white,fill_color=white,time=10483324813.71,edge_size=1
51  DBPost: SIMCOR_RANGE_REPORT  vname=archie,range=181.6211,target=jackal,time=10483324813.705
52  DBPost: SIMCOR_RANGE_REPORT_GT  vname=archie,range=179.033,target=jackal,time=10483324813.705
53  DBPost: VIEW_RANGE_PULSE  x=-186.18,y=-137.77,radius=40,duration=15,label=jackal_ping_reply,
54                            edge_color=chartreuse,fill_color=chartreuse,time=10483324813.71
55  ****************************************************
```

```
56  Range request received from: archie
57     Elapsed time: 31.61322
58     Denied: Range Request exceeds maximum ping frequency.
59  ************************************************************
60  Range request received from: archie
61     Elapsed time: 57.75229
62     Range Request accepted by uSimContactRange.
63     Range Request resets the clock for vehicle.
64     Range Report sent from targ vehicle[jackal] to receiver vehicle[archie]
65  DBPost: VIEW_RANGE_PULSE  x=-140.04,y=-93.47,radius=50,duration=6,fill=,label=archie_ping,
66                            edge_color=white,fill_color=white,time=10483324871.46,edge_size=1
67  DBPost: SIMCOR_RANGE_REPORT  vname=archie,range=132.5609,target=jackal,time=10483324871.457
68  DBPost: SIMCOR_RANGE_REPORT_GT  vname=archie,range=132.2973,target=jackal,time=10483324871.457
69  DBPost: VIEW_RANGE_PULSE  x=-192.85,y=-214.77,radius=40,duration=15,label=jackal_ping_reply,
70                            edge_color=chartreuse,fill_color=chartreuse,time=10483324871.46
71  ********************************************************************************
```

Unless the verbose setting is turned on, the output ending on line 40 above should be the last output written to the console for the duration of the simulator.

In the verbose mode, the simulator will produce event-based output as shown in the example above beginning on line 41. The asterisks in lines 41-43 are not merely visual separators. An asterisk represents a single receipt of a `NODE_REPORT` message. Receiving node reports is essential for the operation of the simulator and this provides a bit of visual verification that this is indeed occurring. Presumably node reports are being received much more often than range requests and range reports are handled, as is the case in the above example. The first time a node report is received for a particular vehicle, rather than an asterisk output, an percent-sign, `'%'`, is output instead, as on line 41 for the two vehicles in this example.

In addition to handling incoming node reports, on any given iteration, the simulator may also handle an incoming range request. Console output for incoming range requests may look like that shown in lines 44-54 above. First the range request and the requesting vehicle is announced as online 44. The elapsed time since the vehicle last made a range request is shown as on line 45. If the request is honored by the simulator, this is indicated as shown on line 46. Otherwise a reason for denial may be shown, as on line 58. If the request is accepted, range reports may be generated for one or more vehicles. For each such vehicle, a line showing the new report is generated, as on line 67.

If artificial noise is being applied by the simulator, e.g., as in line 26 in Listing 46, and the `ground_truth` parameter is set to `true`, then `uSimContactRange` will also generate a "ground truth" report alongside the normal range report. This is indicated in the console output as in lines 52 and 68 above. This ground truth report may not be communicated to the vehicle, but may be used later for post-mission analysis.

14.6 Interaction between uSimContactRange and pMarineViewer

The uSimContactRange application will post certain messages to the MOOSDB that may be subscribed for by GUI based applications like pMarineViewer for visualizing the posting of SIMCOR_RANGE_REPORT and SIMCOR_RANGE_REQUEST messages. A *range pulse* message is used by the uSimContactRange application to convey visually the generation of a range report, or the receipt of a range request. The pulse is rendered as a ring with a growing radius having the vehicle at the center. A pulse emanating By default different colors may be used for range requests and range reports. The RANGE_PULSE message is a data structure implemented in the XYRangePulse class, and usually passed through the MOOS variable VIEW_RANGE_PULSE with the following format:

```
VIEW_RANGE_PULSE = <pulse>
```

The <pulse> component is a series of comma-separated parameter=value pairs. The supported parameters are: x, y, radius, duration, time, fill, fill_color, label, edge_color, and edge_size.

The x and y parameters convey the center of the pulse. The radius parameter indicates the radius of the circle at its maximum. The duration parameter is the number of seconds the pulse will be rendered. The pulse will grow its radius linearly from zero meters at zero seconds to radius meters at duration seconds. The fill parameter is in the range $[0, 1]$, where 0 is full transparency (clear) and 1 is fully opaque. The pulse transparency increases linearly as the range ring is rendered. The starting transparency at $radius = 0$ is given by the fill parameter. The transparency at the maximum radius is zero. The fill_color parameter specifies the color rendered to the internal part of the range pulse. The choice of legal colors is described in Appendix B. The label is a string that uniquely identifies the range instance to consumers like pMarineViewer. As with other objects in pMarineViewer, if it receives an object the same label and type as one previously received, it will replace the old object with the new one in its memory. The edge_color parameter describes the color of the ring rendered in the range pulse. The edge_size likewise describes the line width of the rendered ring. The time parameter indicates the UTC time at which the range pulse object was generated.

Below is an example string representing a range pulse. Each field, with the exception of the x, y position, label, and time, indicates the default values if left unspecified:

```
VIEW_RANGE_PULSE = x=-40,y=-150,radius=40,duration=15,fill=0.25,fill_color=green,label=04
                   edge_color=green,time=3892830128.5,edge_size=1
```

One further note to developers of other apps perhaps wishing to also generate a range pulse for viewing - the recommended manner to generate a range pulse string is to create an instance of the XYRangePulse class using the defined class interface. The string should be obtained by invoking the serialization method for that class. This will better ensure compatibility as the software evolves. The class is defined in lib_geometry in the MOOS-IvP tree.

14.7 The Hugo Example Mission Using uSimContactRange

The *hugo* mission is distributed with the MOOS-IvP source code and contains a ready example of the uSimContactRange application. Assuming the reader has downloaded the source code available at www.moos-ivp.org and built the code according to the discussion in Section 1.4, the *hugo* mission may be launched by:

```
$ cd moos-ivp/ivp/missions/m8_hugo/
$ ./launch.sh 10
```

The argument, 10, in the line above will launch the simulation in `10x` real time. Once this launches, the `pMarineViewer` GUI application should launch and the mission may be initiated by hitting the `DEPLOY` button. Two vehicles should be visibly moving, one surface vehicle labeled `"archie"` and one UUV labeled `"jackal"`, as shown in Figure 36.

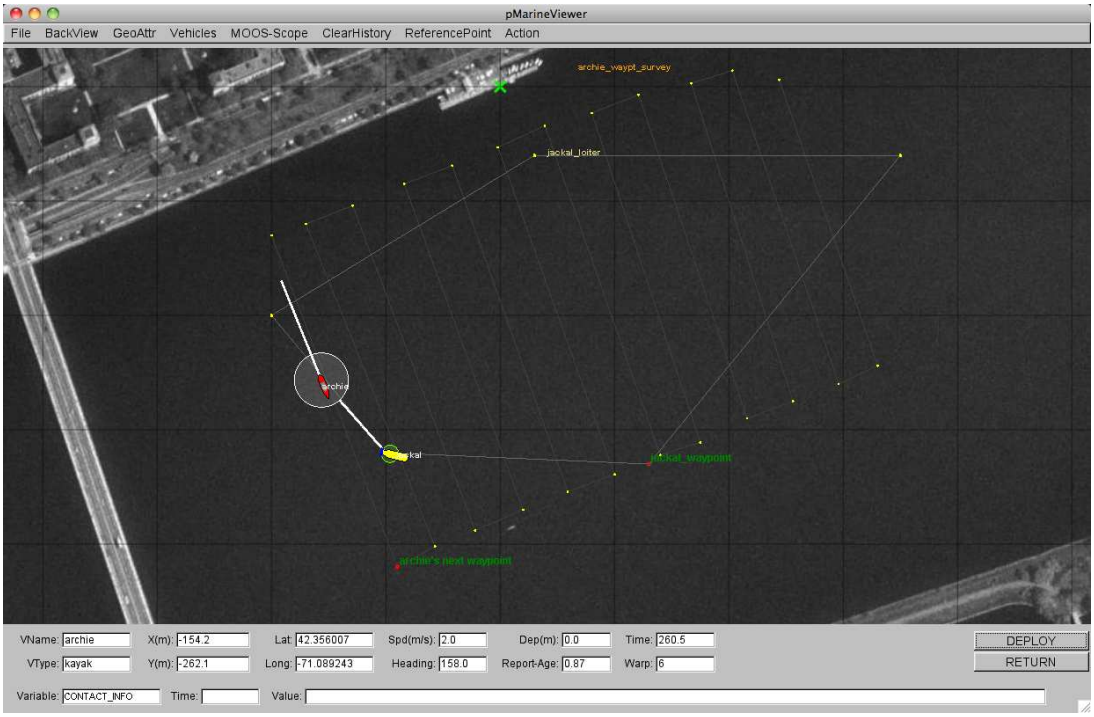


Figure 36: **The Hugo Example Mission:** The Hugo example mission involves two simulated vehicles. The first vehicle is a surface vehicle, *archie*, traversing a lawnmower shaped search pattern. The second vehicle, is a UUV, *jackal*, traversing a polygon pattern overlapping the first pattern. Periodically *archie* emits a ping (range request). This example contains three distinct MOOS communities - one for each simulated vehicle, and one for the shoreside community running `uSimContactRange`.

The surface vehicle will traverse a survey pattern over the region shown in Figure 36, periodically generating a range request (ping). With each range request, a white range pulse should be visible around the vehicle. Almost immediately afterwards, a range report for each neighboring vehicle within range is generated and a green range pulse around each "target" vehicle is rendered. The snapshot in Figure 36 depicts a moment in time where the range request visual pulses are around both vehicles. The larger pulse around the surface vehicle indicates it was generated just prior to the reply pulse around the UUV.

How does the simulated vehicle generate a range request? In practice a user may implement an intelligent module to reason about when to generate requests, but in this case the `uTimerScript` application is used by creating a script that repeats endlessly, generating a range request once every 25-35 seconds. The script is also conditioned on `(NAV_SPEED > 0)`, so the pinging doesn't

start until the vehicle is deployed. The configuration for the script can be seen in the uTimerScript configuration block in the shoreside.moos file. More on the uTimerScript application can be found in Section 9.

Examining the Log Data from the Hugo Mission

After the launch script above has launched the simulation, the script should leave the console user with the option to "Exit and Kill Simulation" by hitting the '2' key. Once the vehicles have been deployed and traversed to one's satisfaction, exit the script. Three log directories should have been created by three separate invocations of the pLogger application in the directory where the simulation was launched. The three directories correspond to the three MOOS communities participating in this simulation, and should have the corresponding prefixes, LOG_ARCHIE, LOG_JACKAL, and LOG_SHORESIDE, with the remainder of the directory names composed from the timestamp of the launch.

Let's take a look at some of the data related to the simulation and uSimContactRange in particular. The uSimContactRange app is part of the shoreside community (see Figure 35). The shoreside log file resides in the LOG_SHORESIDE_* directory, and has the .alog suffix. A dump of the entire file reveals a deluge of information. To look at the information relevant to the uSimContactRange application, the file is pruned with the aloggrep tool described in Section 16.5:

```
$ aloggrep LOG_SHORESIDE_12_7_2011_____08_12_05.alog uSimContactRange uTimerScript
```

This produces a subset of the alog file similar to that shown in Listing 48, showing only log entries made by either the uSimContactRange application, or the uTimerScript application which generated all the range requests as described above.

Even though the below log file represents the logged data from the shoreside MOOS community, it also contains entries of postings bridged from other communities. Upon each range request generated by uTimerScript running on the *archie* vehicle, it its bridged to the shoreside community and logged as SIMCOR_RANGE_REQUEST entries shown below. If the simulator decides to honor the range request, it posts a VIEW_RANGE_PULSE message which can be seen below with the label "archie_ping". If the range request is honored by the simulator it also generates the SIMCOR_RANGE_REPORT message. If the simulator is configured to add random noise to the range measurements, and to report ground truth alongside the noisy measurements, the simulator will also post SIMCOR_RANGE_REPORT_GT message as shown below.

In cases where the range request is *not* honored by the simulator (perhaps because it violated the minimum wait time between pings), the SIMCOR_RANGE_REQUEST message is simply logged by the logger and ignored by the simulator. See the entry at timestamp 104.761 below.

Listing 48 - A subset of the Shoreside log file in the Hugo example mission.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% LOG FILE:       ./LOG_SHORESIDE_12_7_2011_____08_12_05/LOG_SHORESIDE_12_7_2011_____08_12_05.alog
%% FILE OPENED ON  Tue Jul 12 08:12:05 2011
%% LOGSTART               23588509053.6
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
76.758   SIMCOR_RANGE_REQUEST   uTimerScript     name=archie
78.157   VIEW_RANGE_PULSE       uSimContactRange x=-49.31,y=-38.86,radius=50,duration=6,label=archie_ping,...
78.157   SIMCOR_RANGE_REPORT    uSimContactRange vname=archie,range=176.6565,target=jackal,time=23588509131.764
```

```
78.157   VIEW_RANGE_PULSE       uSimContactRange x=-187.67,y=-144.6,radius=40,duration=15,label=jackal_reply,...
78.157   SIMCOR_RANGE_REPORT_GT uSimContactRange vname=archie,range=174.1391,target=jackal,time=23588509131.764
104.761  SIMCOR_RANGE_REQUEST   uTimerScript     name=archie
130.417  SIMCOR_RANGE_REQUEST   uTimerScript     name=archie
132.118  VIEW_RANGE_PULSE       uSimContactRange x=-141.97,y=-94.55,radius=50,duration=6,label=archie_ping,...
132.119  SIMCOR_RANGE_REPORT    uSimContactRange vname=archie,range=131.1348,target=jackal,time=23588509185.725
132.119  SIMCOR_RANGE_REPORT_GT uSimContactRange vname=archie,range=130.8741,target=jackal,time=23588509185.725
132.119  VIEW_RANGE_PULSE       uSimContactRange x=-191.18,y=-215.82,radius=40,duration=15,label=jackal_reply,...
163.599  SIMCOR_RANGE_REQUEST   uTimerScript     name=archie
165.819  VIEW_RANGE_PULSE       uSimContactRange x=-197.98,y=-128.86,radius=50,duration=6,label=archie_ping,...
165.819  SIMCOR_RANGE_REPORT    uSimContactRange vname=archie,range=122.0918,target=jackal,time=23588509219.426
165.819  SIMCOR_RANGE_REPORT_GT uSimContactRange vname=archie,range=127.1016,target=jackal,time=23588509219.426
165.820  VIEW_RANGE_PULSE       uSimContactRange x=-161.42,y=-250.59,radius=40,duration=15,label=jackal_reply,...
190.192  SIMCOR_RANGE_REQUEST   uTimerScript     name=archie
216.311  SIMCOR_RANGE_REQUEST   uTimerScript     name=archie
217.910  VIEW_RANGE_PULSE       uSimContactRange x=-168.48,y=-226,radius=50,duration=6,label=archie_ping,...
217.910  SIMCOR_RANGE_REPORT    uSimContactRange vname=archie,range=96.955,target=jackal,time=23588509271.516
217.910  SIMCOR_RANGE_REPORT_GT uSimContactRange vname=archie,range=98.0072,target=jackal,time=23588509271.516
217.910  VIEW_RANGE_PULSE       uSimContactRange x=-112.19,y=-306.23,radius=40,duration=15,label=jackal_reply,...
245.493  SIMCOR_RANGE_REQUEST   uTimerScript     name=archie
273.729  SIMCOR_RANGE_REQUEST   uTimerScript     name=archie
275.877  VIEW_RANGE_PULSE       uSimContactRange x=-125.38,y=-332.68,radius=50,duration=6,label=archie_ping,...
275.877  SIMCOR_RANGE_REPORT    uSimContactRange vname=archie,range=88.6767,target=jackal,time=23588509329.484
275.877  SIMCOR_RANGE_REPORT_GT uSimContactRange vname=archie,range=85.9737,target=jackal,time=23588509329.484
275.877  VIEW_RANGE_PULSE       uSimContactRange x=-39.7,y=-325.58,radius=40,duration=15,label=jackal_reply,...
```

In this example, the range requests are made by *archie*, and range reports are generated by `uSimContactRange`, and reported back to *archie*, but not the other vehicle. A quick look at the log file for *archie* reveals this, in Listing 49 below. Each range request is followed by a range report unless it violates the minimum ping wait time set in the simulator, which in this case is 30 seconds.

```
$ cd missions/m8_hugo/LOG_ARCHIE_12_7_2011_____08_12_03/
$ aloggrep LOG_ARCHIE_12_7_2011_____08_12_03.alog uSimContactRange uTimerScript
```

Listing 49 - A subset of the Archie log file in the Hugo example mission.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% LOG FILE:       ./LOG_ARCHIE_12_7_2011_____08_12_03/LOG_ARCHIE_12_7_2011_____08_12_03.alog
%% FILE OPENED ON  Tue Jul 12 08:12:03 2011
%% LOGSTART               23588509017.6
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
112.767   SIMCOR_RANGE_REQUEST  uTimerScript     name=archie
114.166   SIMCOR_RANGE_REPORT   uSimContactRange vname=archie,range=176.6565,target=jackal,time=23588509131.764
140.770   SIMCOR_RANGE_REQUEST  uTimerScript     name=archie
166.426   SIMCOR_RANGE_REQUEST  uTimerScript     name=archie
168.127   SIMCOR_RANGE_REPORT   uSimContactRange vname=archie,range=131.1348,target=jackal,time=23588509185.725
199.608   SIMCOR_RANGE_REQUEST  uTimerScript     name=archie
201.828   SIMCOR_RANGE_REPORT   uSimContactRange vname=archie,range=122.0918,target=jackal,time=23588509219.426
226.201   SIMCOR_RANGE_REQUEST  uTimerScript     name=archie
252.319   SIMCOR_RANGE_REQUEST  uTimerScript     name=archie
253.919   SIMCOR_RANGE_REPORT   uSimContactRange vname=archie,range=96.955,target=jackal,time=23588509271.516
281.502   SIMCOR_RANGE_REQUEST  uTimerScript     name=archie
309.738   SIMCOR_RANGE_REQUEST  uTimerScript     name=archie
311.886   SIMCOR_RANGE_REPORT   uSimContactRange vname=archie,range=88.6767,target=jackal,time=23588509329.484
344.566   SIMCOR_RANGE_REQUEST  uTimerScript     name=archie
```

15 The uSimCurrent Utility: Simulating Water Currents

The `uSimCurrent` MOOS application is a newcomer in the toolbox and documentation is thin. Nevertheless it has been tested and used quite a bit and is worth a quick introduction here for those with a need for some ability to simulate water current on unmanned vehicles.

`uSimCurrent` is intended to be used with the `uSimMarine` simulator, by generating force vectors and publishing them to the MOOSDB. The `uSimMarine` simulator has a generic interface to accept externally published force vectors regardless of the source, written to the variables `USM_FORCE_X`, `USM_FORCE_Y`, and `USM_FORCE_VECTOR`, as described in Section 12. The `uSimCurrent` application reads a provided *current field file* containing an association of water current to positions in the water. On iteration of `uSimCurrent`, the vehicle's current position is noted, looked up in the current-field data structure, and a new force vector is posted. The idea is shown in Figure 37.

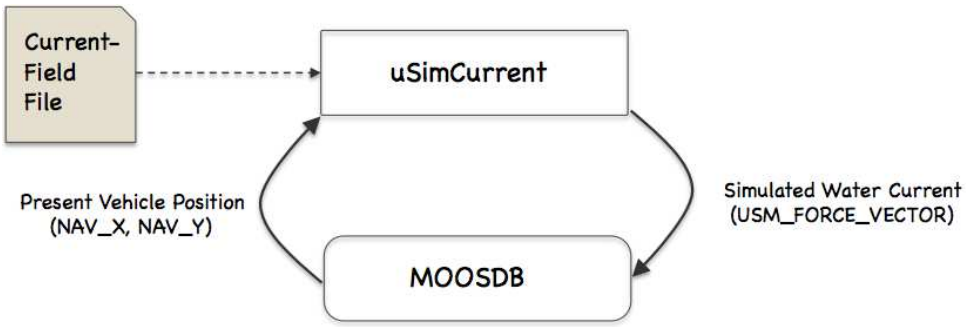


Figure 37: **The `uSimCurrent` Utility:** The simulator is initialized with a data file describing currents and locations. The simulator then repeatedly publishes a current vector based on the present vehicle position.

15.1 Overview of the uSimCurrent Interface and Configuration Options

The `uSimCurrent` 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.

15.1.1 Configuration Parameters for uSimCurrent

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

`CURRENT_FIELD:`	Name of a fild describing a current field.
`CURRENT_FIELD_ACTIVE:`	Boolean indicating whether the simulator is active.

15.1.2 MOOS Variables Posted by uSimCurrent

The primary output of uSimCurrent to the MOOSDB is the force vector to be consumed by the `uSimMarine` application.

`USM_FORCE_VECTOR:`	A force vector representing the prevailing current.
`USC_CFIELD_SUMMARY:`	Summary of configured current field.
`VIEW_VECTOR:`	Vector objects suitable for rendering in GUI applications.

15.1.3 MOOS Variables Subscribed for by uSimCurrent

Variables subscribed for by `uSimCurrent` are summarized below.

`NAV_X:`	The ownship vehicle position on the x axis of local coordinates.
`NAV_Y:`	The ownship vehicle position on the y axis of local coordinates.

If `pNodeReporter` is configured to handle a second navigation solution as described in Section 10.2.5, the corresponding addition variables as described in that section will also be automatically subscribed for.

16 The Alog-Toolbox for Analyzing and Editing Mission Log Files

16.1 Brief Overview

The Alog-Toolbox is a set of five post-mission analysis utility applications `alogview`, `alogscan`, `alogrm`, `aloggrep`, `alogclip`. Each application manipulates or renderings `.alog` files generated by the `pLogger` application. Three of the applications, `alogclip`, `aloggrep`, and `alogrm` are command-line tools for filtering a given `.alog` file to a reduced size. Reduction of a log file size may facilitate the time to load a file in a post-processing application, may facilitate its transmission over slow transmission links when analyzing data between remote users, or may simply ease in the storing and back-up procedures. The `alogscan` tool provides statistics on a given `.alog` file that may indicate how to best reduce file size by eliminating variable entries not used in post-processing. It also generates other information that may be handy in debugging a mission. The `alogview` tool is a GUI-based tool that accepts one or more `.alog` files and renders a vehicle positions over time on an operation area, provides time-correlated plots of any logged numerical MOOS variables, and renders helm autonomy mode data with plots of generated objective functions.

16.2 An Example `.alog` File

The `.alog` file used in the examples below was generated from the Alpha example mission. This file, `alpha.alog`, is found in the missions distributed with the MOOS-IvP tree. The Alpha mission is described in [1]. The `alpha.alog` file was created by simply running the mission as described, and can be found in:

```
moos-ivp/trunk/ivp/missions/alpha/alpha.alog.
```

16.3 The `alogscan` Tool

The `alogscan` tool is a command-line application for providing statistics relating to a given `.alog` file. It reports, for each unique MOOS variable in the log file, (a) the number of lines in which the variable appears, i.e., the number of times the variable was posted by a MOOS application, (b) the total number of characters comprising the variable value for all entries of a variable, (c) the timestamp of the first recorded posting of the variable, (d) the timestamp of the last recorded posting of the variable, (e) the list of MOOS applications the posted the variable.

16.3.1 Command Line Usage for the `alogscan` Tool

The `alogscan` tool is run from the command line with a given `.alog` file and a number of options. The usage options are listed when the tool is launched with the `-h` switch:

Listing 50 - Command line usage for the `alogscan` *tool.*

```
0  > alogscan -h
1  Usage:
2    alogscan file.alog [OPTIONS]
3
4  Synopsis:
5    Generate a report on the contents of a given
6    MOOS .alog file.
7
8  Options:
```

```
 9 --sort=type   Sort by one of SIX criteria:
10                start: sort by first post of a var
11                stop:  sort by last post of a var
12     (Default)  vars:  sort by variable name
13                proc:  sort by process/source name
14                chars: sort by total chars for a var
15                lines: sort by total lines for a var
16
17  --appstat     Output application statistics
18  -r,--reverse  Reverse the sorting output
19  -n,--nocolors Turn off process/source color coding
20  -h,--help     Displays this help message
21  -v,--version  Displays the current release version
22
23  See also: aloggrp, alogrm, alogclip, alogview
```

The order of the arguments passed to `alogscan` do not matter. The lines of output are sorted by grouping variables posted by the same MOOS process or source, as in Listing 51 below. The sorting criteria can instead be done by alphabetical order on the variable name (`--sort=vars`), the total characters in the file due to a variable (`--sort=chars`), the total lines in the file due to a variable (`--sort=lines`), the time of the first posting of the variable (`--sort=start`), or the time of the last posting of the variable (`--sort=stop`). The order of the output may be reversed (`-r, --reverse`). By default, the entries are color-coded by the variable source, using the few available terminal colors (there are not many). When unique colors are exhausted, the color reverts back to the default terminal color in effect at the time.

16.3.2 Example Output from the `alogscan` Tool

The output shown in Listing 51 was generated from the `alpha.alog` file generated by the Alpha example mission.

Listing 51 - Example output from the `alogscan` *tool.*

```
 0  Variable Name           Lines   Chars   Start    Stop  Sources
 1  -------------           -----   -----  ------  ------  -------
 2  DB_CLIENTS                282   22252   -0.38  566.42  MOOSDB_alpha
 3  DB_TIME                   556    7132    1.21  566.18  MOOSDB_alpha
 4  DB_UPTIME                 556    7173    1.21  566.18  MOOSDB_alpha
 5  USIMMARINE_STATUS         276   92705    0.39  565.82  uSimMarine
 6  NAV_DEPTH                6011    6011    1.43  566.38  uSimMarine
 7  NAV_HEADING              6011   75312    1.43  566.38  uSimMarine
 8  NAV_LAT                  6011   74799    1.43  566.38  uSimMarine
 9  NAV_LONG                 6011   80377    1.43  566.38  uSimMarine
10  NAV_SPEED                6011    8352    1.43  566.38  uSimMarine
11  NAV_STATE                6011   18033    1.43  566.38  uSimMarine
12  NAV_X                    6011   72244    1.43  566.38  uSimMarine
13  NAV_Y                    6011   77568    1.43  566.38  uSimMarine
14  NAV_YAW                  6011   80273    1.43  566.38  uSimMarine
15  BHV_IPF                  2009  564165   46.26  542.85  pHelmIvP
16  CREATE_CPU               2108    2348   46.26  566.33  pHelmIvP
17  CYCLE_INDEX                 5       5   44.98  543.09  pHelmIvP
18  DEPLOY                      3      14    3.84  543.09  pHelmIvP,pMarineViewer
19  DESIRED_HEADING          2017    5445    3.85  543.09  pHelmIvP
20  DESIRED_SPEED            2017    2017    3.85  543.09  pHelmIvP
21  HELM_IPF_COUNT           2108    2108   46.26  566.32  pHelmIvP
22  HSLINE                      1       3    3.84    3.84  pHelmIvP
23  IVPHELM_DOMAIN              1      29    3.84    3.84  pHelmIvP
24  IVPHELM_ENGAGED           462    3342    3.85  566.32  pHelmIvP
```

```
25  IVPHELM_MODESET            1       0    3.84    3.84  pHelmIvP
26  IVPHELM_POSTINGS        2014  236320   46.26  543.33  pHelmIvP
27  IVPHELM_STATEVARS          1      20   44.98   44.98  pHelmIvP
28  IVPHELM_SUMMARY         2113  612685   44.98  566.33  pHelmIvP
29  LOOP_CPU                2108    2348   46.26  566.33  pHelmIvP
30  PC_hsline                  1       9   44.98   44.98  pHelmIvP
31  PC_waypt_return            3      14   44.98  543.33  pHelmIvP
32  PC_waypt_survey            3      14   44.98  543.33  pHelmIvP
33  PHELMIVP_STATUS          255  198957    3.85  565.12  pHelmIvP
34  PLOGGER_CMD                1      17    3.84    3.84  pHelmIvP
35  PWT_BHV_HSLINE             1       1   44.98   44.98  pHelmIvP
36  PWT_BHV_WAYPT_RETURN       3       5   44.98  543.09  pHelmIvP
37  PWT_BHV_WAYPT_SURVEY       2       4   44.98  462.90  pHelmIvP
38  RETURN                     4      19    3.84  543.09  pHelmIvP,pMarineViewer
39  STATE_BHV_HSLINE           1       1   44.98   44.98  pHelmIvP
40  STATE_BHV_WAYPT_RETURN     4       4   44.98  543.33  pHelmIvP
41  STATE_BHV_WAYPT_SURVEY     3       3   44.98  463.15  pHelmIvP
42  SURVEY_INDEX              10      10   44.98  429.70  pHelmIvP
43  SURVEY_STATUS           1116   77929   45.97  462.90  pHelmIvP
44  VIEW_POINT              4034  101662   44.98  543.33  pHelmIvP
45  VIEW_SEGLIST               4     273   44.98  543.33  pHelmIvP
46  WPT_INDEX                  1       1  463.15  463.15  pHelmIvP
47  WPT_STAT                 223   15626  463.15  543.09  pHelmIvP
48  LOGGER_DIRECTORY          56    1792    1.07  559.19  pLogger
49  PLOGGER_STATUS           263  331114    1.07  566.40  pLogger
50  DESIRED_RUDDER         10185  150449   -9.28  545.18  pMarinePID
51  DESIRED_THRUST         10637   20774   -9.28  566.52  pMarinePID
52  MOOS_DEBUG                 5      39   -9.31  545.23  pMarinePID,pHelmIvP
53  PMARINEPID_STATUS        279   81990    0.95  566.28  pMarinePID
54  HELM_MAP_CLEAR             1       1   -1.56   -1.56  pMarineViewer
55  MOOS_MANUAL_OVERIDE        1       5   44.65   44.65  pMarineViewer
56  PMARINEVIEWER_STATUS     270   95560   -0.95  564.89  pMarineViewer
57  NODE_REPORT_LOCAL       1159  207535    1.15  565.91  pNodeReporter
58  PNODEREPORTER_STATUS     233   50534   -0.37  563.93  pNodeReporter
59  ------------------------------------------------------------------
60  Total variables: 57
61  Start/Stop Time: -9.31 / 566.52
```

When the `-appstat` command line option is included, a second report is generated, after the above report, that provides statistics keyed by application, rather than by variable. For each application that has posted a variable recorded in the given `.alog` file, the number of lines and characters are recorded, as well as the percentage of total lines and characters. An example for the `alpha.alog` file is shown in Listing 52.

Listing 52 - Example `alogscan` *output generated with the* `-appstat` *command line option.*

```
64  MOOS Application     Total Lines  Total Chars  Lines/Total  Chars/Total
65  ---------------      -----------  -----------  -----------  -----------
66  MOOSDB_alpha               1394        36557         1.37         1.08
67  uSimMarine                54375       585674        53.57        17.29
68  pHelmIvP                  22642      1825437        22.31        53.89
69  pLogger                     319       332906         0.31         9.83
70  pMarinePID                21106       253252        20.80         7.48
71  pMarineViewer               279        95599         0.27         2.82
72  pNodeReporter              1392       258069         1.37         7.62
```

Further Tips

- If a small number of variables are responsible for a relatively large portion of the file size, and are expendable in terms of how data is being analyzed, the variables may be removed to ease

the handling, transmission, or storage of the data. To remove variables from existing files, the `alogrm` tool described in 16.6 may be used. To remove the variable from future files, the `pLogger` configuration may be edited by either removing the variable from the list of variables explicitly requested for logging, or if WildCardLogging is used, mask out the variable with the `WildCardOmitPattern` parameter setting. See the `pLogger` documentation.

- The output of `alogscan` can be further distilled using common tools such as `grep`. For example, if one only wants a report on variables published by the `pHelmIvP` application, one could type:

  ```
  alogscan alpha.alog | grep pHelmIvP
  ```

16.4 The `alogclip` Tool

The `alogclip` tool will prune a given `.alog` file based on a given beginning and end timestamp. This is particularly useful when a log file contains a sizeable stretch of data logged after mission completion, such as data being recorded while the vehicle is being recovered or sitting idle topside after recovery.

16.4.1 Command Line Usage for the `alogclip` Tool

The `alogclip` tool is run from the command line with a given `.alog` file, a start time, end time, and the name of a new `.alog` file. By default, if the named output file exists, the user will be prompted before overwriting it. The user prompt can be bypassed with the `-f,--force` option. The usage options are listed when the tool is launched with the `-h` switch:

Listing 53 - Command line usage for the `alogclip` *tool.*

```
 0  > alogclip -h
 1  Usage:
 2    alogclip in.alog mintime maxtime [out.alog] [OPTIONS]
 3
 4  Synopsis:
 5    Create a new MOOS .alog file from a given .alog file
 6    by removing entries outside a given time window.
 7
 8  Standard Arguments:
 9    in.alog  - The input logfile.
10    mintime  - Log entries with timestamps below mintime
11               will be excluded from the output file.
12    maxtime  - Log entries with timestamps above mintime
13               will be excluded from the output file.
14    out.alog - The newly generated output logfile. If no
15               file provided, output goes to stdout.
16
17  Options:
18    -h,--help     Display this usage/help message.
19    -v,--version  Display version information.
20    -f,--force    Overwrite an existing output file
21    -q,--quiet    Verbose report suppressed at conclusion.
22
23  Further Notes:
24    (1) The order of arguments may vary. The first alog
25        file is treated as the input file, and the first
26        numerical value is treated as the mintime.
27    (2) Two numerical values, in order, must be given.
28    (3) See also: alogscan, alogrm, aloggrep, alogview
```

16.4.2 Example Output from the `alogclip` Tool

The output shown in Listing 54 was generated from the `alpha.alog` file generated by the Alpha example mission.

Listing 54 - Example `alogclip` *output applied to the* `alpha.alog` *file.*

```
 1  > alogclip alpha.alog new.alog 50 350
 2
 3  Processing input file alpha.alog...
 4
 5  Total lines clipped:      44,988  (44.32 pct)
 6    Front lines clipped:     5,474
 7    Back  lines clipped:    39,514
 8  Total chars clipped:   4,200,260  (43.09 pct)
 9    Front chars clipped:   432,409
10    Back  chars clipped: 3,767,851
```

16.5 The `aloggrep` Tool

The `aloggrep` tool will prune a given `.alog` file by retaining lines of the original file that contain log entries for a user-specified list of MOOS variables or MOOS processes (sources). As the name implies it is motivated by the Unix `grep` command, but `grep` will return a matched line regardless of where the pattern appears in the line. Since MOOS variables also often appear in the string content of other MOOS variables, `grep` often returns much more than one is looking for. The `aloggrep` tool will only pattern-match on the second column of data (the MOOS variable name), or the third column of data (the MOOS source), of any given entry in a given `.alog` file.

16.5.1 Command Line Usage for the `aloggrep` Tool

Listing 55 - Command line usage for the `aloggrep` *tool.*

```
 0  > aloggrep -h
 1  Usage:
 2    aloggrep in.alog [VAR] [SRC] [out.alog] [OPTIONS]
 3
 4  Synopsis:
 5    Create a new MOOS .alog file by retaining only the
 6    given MOOS variables or sources from a given .alog file.
 7
 8  Standard Arguments:
 9    in.alog  - The input logfile.
10    out.alog - The newly generated output logfile. If no
11               file provided, output goes to stdout.
12    VAR      - The name of a MOOS variable
13    SRC      - The name of a MOOS process (source)
14
15  Options:
16    -h,--help     Displays this help message
17    -v,--version  Displays the current release version
18    -f,--force    Force overwrite of existing file
19    -q,--quiet    Verbose report suppressed at conclusion
20
21  Further Notes:
22    (1) The second alog is the output file. Otherwise the
23        order of arguments is irrelevent.
```

```
24    (2) VAR* matches any MOOS variable starting with VAR
25    (3) See also: alogscan, alogrm, alogclip, alogview
```

Note that, in specifying items to be filtered out, there is no distinction made on the command line that a given item refers to a entry's variable name or an entry's source, i.e., MOOS process name.

16.5.2 Example Output from the `aloggrep` Tool

The output shown in Listing 56 was generated from the `alpha.alog` file generated by the Alpha example mission.

Listing 56 - Example `aloggrep` *output applied to the* `alpha.alog` *file.*

```
 1  > aloggrep alpha.alog NAV_* new.alog
 2
 3  Processing on file : alpha.alog
 4    Total lines retained: 54099 (53.30%)
 5    Total lines excluded: 47396 (46.70%)
 6    Total chars retained: 3293774 (33.79%)
 7    Total chars excluded: 6453494 (66.21%)
 8      Variables retained: (9) NAV_DEPTH, NAV_HEADING, NAV_LAT, NAV_LONG,
 9  NAV_SPEED, NAV_STATE, NAV_X, NAV_Y, NAV_YAW
```

16.6 The `alogrm` Tool

The `alogrm` tool will prune a given `.alog` file by removing lines of the original file that contain log entries for a user-specified list of MOOS variables or MOOS processes (sources). It may be fairly viewed as the complement of the `aloggrep` tool.

16.6.1 Command Line Usage for the `alogrm` Tool

Listing 57 - Command line usage for the `alogrm` *tool.*

```
 0  > alogrm -h
 1  Usage:
 2    alogrm in.alog [VAR] [SRC] [out.alog] [OPTIONS]
 3
 4  Synopsis:
 5     Remove the entries matching the given MOOS variables or sources
 6     from the given .alog file and generate a new .alog file.
 7
 8  Standard Arguments:
 9    in.alog  - The input logfile.
10    out.alog - The newly generated output logfile. If no
11               file provided, output goes to stdout.
12    VAR      - The name of a MOOS variable
12    SRC      - The name of a MOOS process (source)
14
15  Options:
16    -h,--help     Displays this help message
17    -v,--version  Displays the current release version
18    -f,--force    Force overwrite of existing file
19    -q,--quiet    Verbose report suppressed at conclusion
20    --nostr       Remove lines with string data values
21    --nonum       Remove lines with double data values
22    --clean       Remove lines that have a timestamp that is
```

```
23                 non-numerical or lines w/ no 4th column
24
25  Further Notes:
26    (1) The second alog is the output file. Otherwise the
27        order of arguments is irrelevent.
28    (2) VAR* matches any MOOS variable starting with VAR
29    (3) See also: alogscan, aloggrep, alogclip, alogview
```

Note that, in specifying items to be filtered out, there is no distinction made on the command line that a given item refers to a entry's variable name or an entry's source, i.e., MOOS process name.

16.6.2 Example Output from the `alogrm` Tool

The output shown in Listing 58 was generated from the `alpha.alog` file generated by the Alpha example mission.

Listing 58 - Example `alogrm` *output applied to the* `alpha.alog` *file.*

```
 1  > alogrm alpha.alog NAV_* new.alog
 2
 3  Processing on file : alpha.alog
 4
 5    Total lines retained: 47396 (46.70%)
 6    Total lines excluded: 54099 (53.30%)
 7    Total chars retained: 6453494 (66.21%)
 8    Total chars excluded: 3293774 (33.79%)
 9    Variables retained: (48) BHV_IPF, CREATE_CPU, CYCLE_INDEX, DB_CLIENTS,
10  DB_TIME, DB_UPTIME, DEPLOY, DESIRED_HEADING, DESIRED_RUDDER, DESIRED_SPEED,
11  DESIRED_THRUST, HELM_IPF_COUNT, HELM_MAP_CLEAR, HSLINE, USIMMARINE_STATUS,
12  IVPHELM_DOMAIN, IVPHELM_ENGAGED, IVPHELM_MODESET, IVPHELM_POSTINGS,
13  IVPHELM_STATEVARS, IVPHELM_SUMMARY, LOGGER_DIRECTORY, LOOP_CPU, MOOS_DEBUG,
14  MOOS_MANUAL_OVERIDE, NODE_REPORT_LOCAL, PC_hsline, PC_waypt_return,
15  PC_waypt_survey, PHELMIVP_STATUS, PLOGGER_CMD, PLOGGER_STATUS,
16  PMARINEPID_STATUS, PMARINEVIEWER_STATUS, PNODEREPORTER_STATUS,
17  PWT_BHV_HSLINE, PWT_BHV_WAYPT_RETURN, PWT_BHV_WAYPT_SURVEY, RETURN,
18  STATE_BHV_HSLINE, STATE_BHV_WAYPT_RETURN, STATE_BHV_WAYPT_SURVEY,
19  SURVEY_INDEX, SURVEY_STATUS, VIEW_POINT, VIEW_SEGLIST, WPT_INDEX, WPT_STAT
```

16.7 The `alogview` Tool

The `alogview` application is used for post-mission rendering of one or more alog files. It provides (a) an indexed view of vehicle position rendered on the operation area, (b) time plots of any logged numerical data, (c) IvP Helm information for a given point in time, and (d) rendered IvP functions generated by the helm for a given point in time. A snapshot of the tool is shown in Figure 38.

This tool is very much still under development, and the below documentation is far from complete despite the best intentions after release 4.1. Nevertheless, since there are those who are using this regularly at this date, some attempt here is made to introduce the tool. This tool was also known as `logview` prior to release 4.1 in August 2010. It was renamed to `alogview` to make it consistent with other tools in the Alog Toolbox. A significant addition to the tool since Release 4.1 is the support for rendering depth objective functions as shown in the figure.

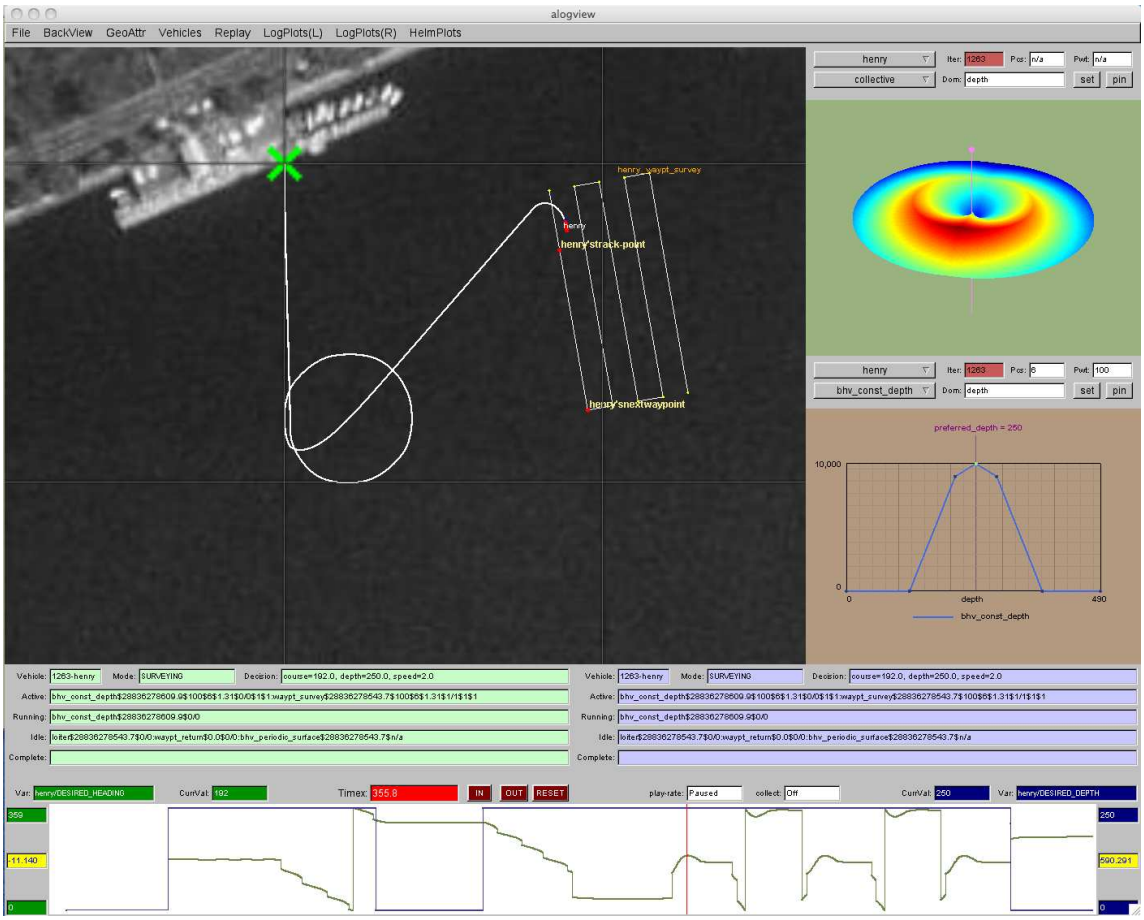


Figure 38: **The `alogview` tool**: used for post-mission rendering of alog files from one or more vehicles and stepping through time to analyze helm status, IvP objective functions, and other logged numerical data correlated to vehicle position in the op-area and time.

The view shown to the user at any given time is indexed on a timestamp. The vehicles rendered

in the op-area are shown at their positions for that point in time. The helm scope output and IvP function output are displayed for the helm iteration at the current timestamp. The data plot output shows a plot for a given variable over all logged time with a red vertical bar that moves left to right indicating the current time. The `alogview` tool by default loads the complete alog vehicle history to allow the user to jump directly to any point in time. The option also exists to load only a portion of the data based on start and end time provided on the command line.

16.7.1 Command Line Usage for the `alogview` Tool

The `alogview` tool is run from the command line with one or more given `.alog` files and a number of options. The usage options are listed when the tool is launched with the `-h` switch:

Listing 59 - Command line usage for the `alogview` *tool.*

```
 0  > alogview -h
 1  Usage:
 2    alogview file.alog [another_file.alog] [OPTIONS]
 3
 4  Synopsis:
 5    Renders vehicle paths from multiple MOOS .alog files.
 6    Renders time-series plots for any logged numerical data.
 7    Renders IvP Helm mode information vs. vehicle position.
 8    Renders IvP Helm behavior objective functions.
 9
10  Standard Arguments:
11    file.alog - The input logfile.
12
13  Options:
14    -h,--help      Displays this help message
15    -v,--version   Displays the current release version
16    --mintime=val  Clip data with timestamps < val
17    --maxtime=val  Clip data with timestamps > val
18    --nowtime=val  Set the initial startup time
19    --geometry=val Viewer window pixel size in HEIGHTxWIDTH
20                   or large, medium, small, xsmall
21                   Default size is 1400x1100 (large)
22    --layout=val   Window layout=normal,noipfs, or fullview
23
24  Further Notes:
25    (1) Multiple .alog files ok - typically one per vehicle
26    (2) Non alog files will be scanned for polygons
27    (3) See also: alogscan, alogrm, alogclip, aloggrep
```

The order of the arguments passed to `alogview` do not matter. The `--mintime` and `--maxtime` arguments allow the user to effectively clip the alog files to reduce the amount of data loaded into RAM by `alogview` during a session. The `--geometry` argument allows the user to custom set the size of the display window. A few shortcuts, `"large"`, `"medium"`, `"small"`, and `"xsmall"` are allowed. The `--layout` argument allows the user to affect the real estate layout by optionally closing one or more panels to enlarge other panels. This is described in Section 16.7.2.

16.7.2 Description of Panels in the `alogview` Window

Although the `alogview` tool will read in any `.alog` file produced by the `pLogger` tool, much of the tool's screen real estate is dedicated to rendering information produced by the helm. The `alogview` tool has six panels of information, as shown in Figure 39. The primary panel is the Op-Area Panel which renders the vehicle position(s) on the operation area as a function of time, along with track

history. The IvPFunction Panels render the objective functions produced by vehicle behaviors for a given helm iteration. The Helm Scope Panels display helm output and behavior information for a given helm iteration. The Data Plot panels render two plots of logged numerical data vs the vehicle log time. Each of these panels and panel controls are discussed in the next few sections.

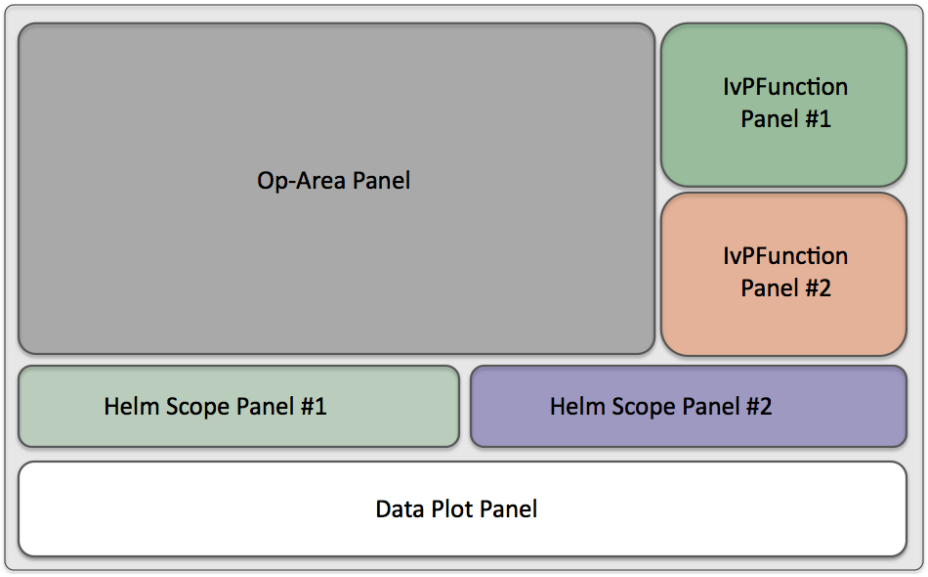


Figure 39: **The panels comprising the `alogview` tool**: Each panel renders information based on the globally held current timestamp. Certain panels may be collapsed to make more room for other panels.

16.7.3 The Op-Area Panel for Rendering Vehicle Trajectories

The Op-Area panel renders, for a given point in time, the vehicle(s) current position and orientation, the vehicle(s) trajectory history, and certain geometric objects such as points, polygons, line segments and their labels, that may have been posted by the helm or other MOOS processes if they were logged in the alog file(s). An example is shown in Figure 40.

Stepping Through Time - The Playback Pull-down Menu

The primary interaction with the Op-Area panel is to step the vehicle forward and backward through time either by fixed time increments or by helm iteration. The simplest way to step is with either the `'['` and `']'` keys to step backward and forward by one step, or the `'<'` and `'>'` keys to move by ten steps. (The current time can also be jumped to with a mouse click in the Data Plot panel. A step unit by default is one second. Alternatively the step unit may be given by one iteration of the helm. If multiple vehicles (alog files) are open, a helm iteration is defined by the helm in the "active" vehicle, i.e, the vehicle whose helm scope information is being displayed in the left-hand Helm Scope panel.

The stepping can be initiated by successive key clicks as noted above or be automatic when put into playback, i.e., streaming mode. In streaming mode the steps continue automatically at

Figure 40: **The OpArea panel of the `alogview` tool**: Vehicle position(s) for a given point in time are rendered along with vehicle trajectory history and certain geometric visual artifacts such as points, polygons and line segments that may have been posted by the helm or another MOOS process. The image in this figure can be replicated exactly by launching `alogview` on the `alpha.alog` file from the Alpha mission distributed with the MOOS-IvP source code, launched with the `--nowtime=144` command line option.

fixed intervals until paused or until the end of the latest timestamp of all loaded log files. The time interval between step executions can be sped up or slowed down with the `'a'` or `'z'` keys respectively. The primary motivation of streaming was to have the option of doing a screen capture of images on each step saved to a file for later compilation as an animation (typically animated GIF). Screen capturing can be enabled from the Playback pull-down menu or by hitting the `'w'` key. When enabled, a purple box should be rendered over the Op-Area panel indicating the scope of the screen capture. By successively hitting the `'w'` key, the capture box is changed between the following extents: `"1024x768"`, `"800x600"`, `"640x480"`, `"480x360"`, and `"off"`. The capturing is done by invoking a system call to the `import` tool distributed with the powerful ImageMagick Open Source package.

Vehicle and Vehicle History Renderings - The Vehicles Pull-down Menu

The rendering of vehicle size, type and color, vehicle trails and the position of the vehicles may be altered via the Vehicles pull-down menu options. The vehicle size and shape upon startup is determined from the `NODE_REPORT_LOCAL` variable (See Section 10.1.2). The set of displayable shapes in `alogview` is the same as that for the `pMarineViewer` application and shown in Figure 10 on page 33. The rendering of vehicles in the Op-Area panel may be toggled off and on by hitting the `CTRL-'v'` key, and the size of the vehicles may be altered with the `'-'` and `'+'`. The size of the rendered vehicle initially is drawn to scale based on the length reported in `NODE_REPORT_LOCAL`, and can be returned to scale by hitting the `ALT-'v'` key.

The rendering of vehicle names may be toggled off and on by hitting the 'n' key, and the user may toggle between a few different choices for text color by hitting the ALT-'n' key. By default the color of the vehicles is set to be yellow for all *inactive* vehicles, and red for the one *active* vehicle (where *active* means it is the vehicle whose helm data is shown in the left-hand Helm Scope panel). A few different choices for active and inactive vehicle colors are provided in the Vehicles pull-down menu. The selection of the active vehicle can be made explicitly from the HelmPlots pull-down menu, by cycling through the vehicles by hitting the 'v' key.

Vehicle trails, i.e., position history, are by default rendered from the present point in time back to the beginning of logged positions in the alog file. Three other modes are supported and can be toggled through with the 't' key. The other modes are: no trails shown at all, all trails shown from the start to end log time, and trails with a limited history. In the latter case the trail stays a fixed length behind the vehicle. This fixed length can be made shorter or longer with the '(' or ')' keys respectively. The size of each trail point can be made smaller or larger with the '[' or ']' keys respectively.

Geometric Object Renderings - The GeoAttr Pull-down Menu

Certain geometric objects logged in the alog file may be displayed in the Op-Area panel and their renderings affected by choices available in the GeoAttr pull-down menu. Each object is posted with a tag that allows for the object to be effectively erased when an object of the same type and tag is subsequently posted. The alogview tool, upon startup, reads through the log file(s) and determines which geometric objects are viewable at any given point in time. Thus the user may see these objects disappear and reappear as one steps back and forth through time. Current objects supported by alogview are VIEW_POINT, VIEW_SEGLIST, VIEW_POLYGON, VIEW_MARKER, and VIEW_RANGE_PULSE

16.7.4 The Helm Scope Panels for View Helm State by Iteration

The *helm scope* panels are used for examining the state of the vehicle helm at the present point in time. In the Vehicle: box, the name of the vehicle preceded by the helm iteration is shown. In the Mode: box, the helm's current mode is given if hierarchical mode declarations are configured for the helm. In the Decision: box, the helm's decision for that iteration is shown for each decision variable. The Active: box, shows the list of behaviors active on the present helm iteration. The Running: box, shows the list of behaviors running on the present helm iteration. The Idle: box, shows the list of behaviors idle on the present helm iteration. The Completed: box, shows the list of behaviors that have been completed as of the present helm iteration.

If there are multiple vehicles (alog files) being viewed, the user can switch the vehicle for each helm scope panel via the HelmPlots pull-down menu.

16.7.5 The Data Plot Panel for Logged Data over Time

The *data plot* panels at the bottom of the window allow the user to plot any two logged MOOS variables, if they were logged as numerical data. A red bar in the time plot indicates the current point in time so the user can visually correlate the vehicles' position in the op area relative to the data plot. The user can also click anywhere on the time plot to alter the current point in time used in all panels. The user may also zoom in on the data plot for better resolution. One note of

caution - the scales used by the two variables are likely not the same. The range from low to high for the particular variable. The range of values is shown on the far left and far right.

16.7.6 Automatic Replay of the Log file(s)

The user can allow the `alogview` tool to step through time automatically, effectively replaying the mission over its duration. Replaying can be adjusted in the Replay pull-down menu. The `'='` key toggles replaying, and the `'a'` and `'z'` keys can be used to slow down or speed up the replay rate. This feature is useful when used with other software that allows automatic generation of video capture real-time from the display. QuickTime in OS X for example.

A Use of Logic Expressions

Logic conditions are employed in both the `pHelmIvP` and `uTimerScript` applications, to condition certain activities based on the prescribed logic state of elements of the MOOSDB. The use of logic conditions in the helm is done in behavior file (`.bhv` file). For the `uTimerScript` application, logic conditions are used in the configuration block of the mission file (`.moos` file). The MOOS application using logic conditions maintains a local buffer representing a snapshot of the MOOSDB for variables involved in the logic expressions. The key relationships and steps are shown in Figure 41:

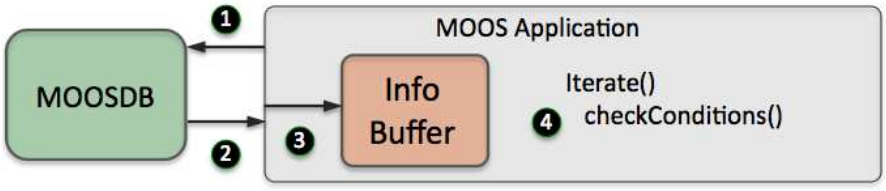


Figure 41: **Logic conditions in a MOOS application**: Step 1: the applications registers to the MOOSDB for any MOOS variables involved in the logic expressions. Step 2: The MOOS application reads incoming mail from the MOOSDB. Step 3: Any new mail results in an update to the information buffer. Step 4: Within the applications Iterate() method, the logic expressions are evaluated based on the contents of the information buffer.

The logic conditions are configured as follows:

```
CONDITION = <logic-expression>
```

The keyword `CONDITION` and is case insensitive. When multiple conditions are specified, it is implied that the overall criteria for "meeting conditions" is the conjunction of all such conditions. In what remains below, the allowable syntax for `<logic-expression>` is described.

Simple Relational Expressions

Each logic expression is comprised of either Boolean operators (and, or, not) or relation operators ($\leq, <, \geq, >, =, !=$). All expressions have at least one relational expression, where the left-hand side of the expression is treated as a variable, and the right-hand side is a literal (either a string or numerical value). The literals are treated as a string value if quoted, or if the value is non-numerical. Some examples:

```
DEPLOY   = true    // Example 1

QUALITY >= 75      // Example 2
```

Variable names are case sensitive since MOOS variables in general are case sensitive. In matching string values of MOOS variables in Boolean conditions, the matching is *case insensitive*. If for example, in Example 1 above, the MOOS variable `DEPLOY` had the value `"TRUE"`, this would satisfy the condition. But if the MOOS variable `deploy` had the value `"true"`, this would not satisfy Example 1.

Simple Logical Expressions with Two MOOS Variables

A relational expression generally involves a variable and a literal, and the form is simplified by insisting the variable is on the left and the literal on the right. A relational expression can also involve the comparison of two variables by surrounding the right-hand side with `$()`. For example:

```
REQUESTED_STATE != $(RUN_STATE)                // Example 3
```

The variable types need to match or the expression will evaluate to `false` regardless of the relation. The expression in Example 3 will evaluate to `false` if, for example, `REQUESTED_STATE="run"` and `RUN_STATE=7`, simply because they are of different type, and regardless of the relation being the inequality relation.

Complex Logic Expressions

Individual relational expressions can be combined with Boolean connectors into more complex expressions. Each component of a Boolean expression must be surrounded by a pair of parentheses. Some examples:

```
(DEPLOY = true) or (QUALITY >= 75)              // Example 4

(MSG != error) and !((K <= 10) or (w != 0))     // Example 5
```

A relational expression such as `(w != 0)` above is false if the variable `w` is undefined. In MOOS, this occurs if variable has yet to be published with a value by any MOOS client connected to the `MOOSDB`. A relational expression is also `false` if the variable in the expression is the wrong type, compared to the literal. For example `(w != 0)` in Example 5 would evaluate to `false` even if the variable `w` had the string value `"alpha"` which is clearly not equal to zero.

B Colors

Below are the colors used by IvP utilities that use colors. Colors are case insensitive. A color may be specified by the string as shown, or with the '_' character as a separator. Or the color may be specified with its hexadecimal or floating point form. For example the following are equivalent: "darkblue", "DarkBlue", "dark_blue", "hex:00,00,8b", and "0,0,0.545". In the latter two styles, the '%', '$', or '#' characters may also be used as a delimiter instead of the comma if it helps when embedding the color specification in a larger string that uses its own delimeters. Mixed delimeters are not supported however.

antiquewhite, (fa,eb,d7)
aqua (00,ff,ff)
aquamarine (7f,ff,d4)
azure (f0,ff,ff)
beige (f5,f5,dc)
bisque (ff,e4,c4)
black (00,00,00)
blanchedalmond(ff,eb,cd)
blue (00,00,ff)
blueviolet (8a,2b,e2)
brown (a5,2a,2a)
burlywood (de,b8,87)
cadetblue (5f,9e,a0)
chartreuse (7f,ff,00)
chocolate (d2,69,1e)
coral (ff,7f,50)
cornsilk (ff,f8,dc)
cornflowerblue(64,95,ed)
crimson (de,14,3c)
cyan (00,ff,ff)
darkblue (00,00,8b)
darkcyan (00,8b,8b)
darkgoldenrod (b8,86,0b)
darkgray (a9,a9,a9)
darkgreen (00,64,00)
darkkhaki (bd,b7,6b)
darkmagenta (8b,00,8b)
darkolivegreen(55,6b,2f)
darkorange (ff,8c,00)
darkorchid (99,32,cc)
darkred (8b,00,00)
darksalmon (e9,96,7a)
darkseagreen (8f,bc,8f)
darkslateblue (48,3d,8b)
darkslategray (2f,4f,4f)
darkturquoise (00,ce,d1)
darkviolet (94,00,d3)
deeppink (ff,14,93)
deepskyblue (00,bf,ff)
dimgray (69,69,69)
dodgerblue (1e,90,ff)
firenrick (b2,22,22)
floralwhite (ff,fa,f0)
forestgreen (22,8b,22)
fuchsia (ff,00,ff)
gainsboro (dc,dc,dc)
ghostwhite (f8,f8,ff)
gold (ff,d7,00)
goldenrod (da,a5,20)
gray (80,80,80)
green (00,80,00)
greenyellow (ad,ff,2f)
honeydew (f0,ff,f0)
hotpink (ff,69,b4)
indianred (cd,5c,5c)
indigo (4b,00,82)
ivory (ff,ff,f0)
khaki (f0,e6,8c)
lavender (e6,e6,fa)
lavenderblush (ff,f0,f5)
lawngreen (7c,fc,00)
lemonchiffon (ff,fa,cd)
lightblue (ad,d8,e6)
lightcoral (f0,80,80)
lightcyan (e0,ff,ff)
lightgoldenrod(fa,fa,d2)
lightgray (d3,d3,d3)
lightgreen (90,ee,90)

lightpink (ff,b6,c1)
lightsalmon (ff,a0,7a)
lightseagreen (20,b2,aa)
lightskyblue (87,ce,fa)
lightslategray(77,88,99)
lightsteelblue(b0,c4,de)
lightyellow (ff,ff,e0)
lime (00,ff,00)
limegreen (32,cd,32)
linen (fa,f0,e6)
magenta (ff,00,ff)
maroon (80,00,00)
mediumblue (00,00,cd)
mediumorchid (ba,55,d3)
mediumseagreen(3c,b3,71)
mediumslateblue(7b,68,ee)
mediumspringgreen(00,fa,9a)
mediumturquoise(48,d1,cc)
mediumvioletred(c7,15,85)
midnightblue (19,19,70)
mintcream (f5,ff,fa)
mistyrose (ff,e4,e1)
moccasin (ff,e4,b5)
navajowhite (ff,de,ad)
navy (00,00,80)
oldlace (fd,f5,e6)
olive (80,80,00)
olivedrab (6b,8e,23)
orange (ff,a5,00)
orangered (ff,45,00)
orchid (da,70,d6)
palegreen (98,fb,98)
paleturquoise (af,ee,ee)
palevioletred (db,70,93)
papayawhip (ff,ef,d5)
peachpuff (ff,da,b9)
pelegoldenrod (ee,e8,aa)
peru (cd,85,3f)
pink (ff,c0,cb)
plum (dd,a0,dd)
powderblue (b0,e0,e6)
purple (80,00,80)
red (ff,00,00)
rosybrown (bc,8f,8f)
royalblue (41,69,e1)
saddlebrowm (8b,45,13)
salmon (fa,80,72)
sandybrown (f4,a4,60)
seagreen (2e,8b,57)
seashell (ff,f5,ee)
sienna (a0,52,2d)
silver (c0,c0,c0)
skyblue (87,ce,eb)
slateblue (6a,5a,cd)
slategray (70,80,90)
snow (ff,fa,fa)
springgreen (00,ff,7f)
steelblue (46,82,b4)
tan (d2,b4,8c)
teal (00,80,80)
thistle (d8,bf,d8)
tomatao (ff,63,47)
turquoise (40,e0,d0)
violet (ee,82,ee)
wheat (f5,de,b3)
white (ff,ff,ff)
whitesmoke (f5,f5,f5)
yellow (ff,ff,00)
yellowgreen (9a,cd,32)

References

[1] Michael R. Benjamin, Paul M. Newman, Henrik Schmidt, and John J. Leonard. An Overview of MOOS-IvP and a Brief Users Guide to the IvP Helm Autonomy Software. Technical Report MIT-CSAIL-TR-2009-028, MIT Computer Science and Artificial Intelligence Lab, June 2009.

[2] Michael R. Benjamin, Paul M. Newman, Henrik Schmidt, and John J. Leonard. An Overview of MOOS-IvP and a Users Guide to the IvP Helm Autonomy Software. Technical Report MIT-CSAIL-TR-2010-041, MIT Computer Science and Artificial Intelligence Lab, August 2010.

[3] Mary M. Hunt, William M. Marquet, Donald A. Moller, Kenneth R. Peal, Woollcott K. Smith, and Rober C. Spindel. An Acoustic Navigation System. Technical Report WHOI-74-6, Woods Hole Oceanographic Institution, Woods Hole, Massachusetts, December 1974.

[4] P. A. Milne. *Underwater Acoustic Positioning Systems*. Gulf Publishing Co., Houston TX, January 1983.

[5] Roger P. Sokey and Thomas C. Austin. Sequential Long-Baseline Navigation for REMUS, an Autonomous Underwater Vehicle. *Proceedings of SPIE, the Internation Society for Optical Engineering*, 3711:212–219a, 1999.

[6] Louis L. Whitcomb, Dana R. Yoerger, and Hanumant Singh. Combined Doppler/LBL Based Navigation of Underwater Vehicles. In *11th International Symposium on Unmanned Untethered Submersible Technology (UUST99)*, Durham, New Hampshire, August 1999.

Index

Index of MOOS Variables