Computer Science and Artificial Intelligence Laboratory
Technical Report

MIT-CSAIL-TR-2008-065 November 11, 2008

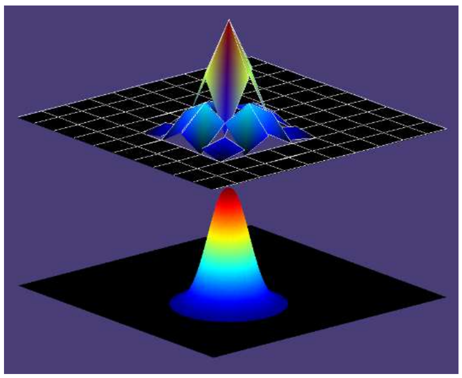
MOOS-IvP Autonomy Tools Users Manual

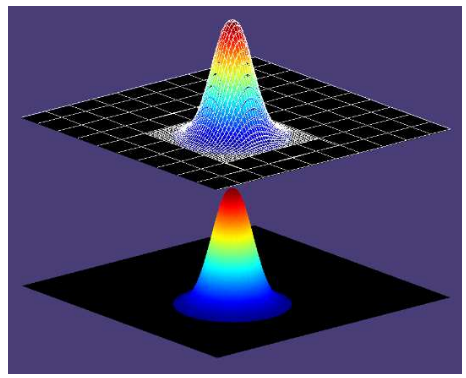
Michael R. Benjamin

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

MOOS-IvP Autonomy Tools Users Manual

The uHelmScope, pMarineViewer, uXMS, uTermCommand, pEchoVar, uProcessWatch and uPokeDB Tools





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

Center for Advanced System Technologies
NUWC Division Newport, Newport RI

November 4th, 2008

Abstract

This document describes seven common MOOS-IvP autonomy tools. The uHelmScope application provides a run-time scoping window into the state of an active IvP Helm executing its mission. The pMarineViewer application is a geo-based GUI tool for rendering marine vehicles and certain autonomy properties in their operational area. The uXMS application is a terminal based tool for live scoping on a MOOSDB process. The uTermCommand application is a terminal based tool for poking the MOOSDB with a set of MOOS file pre-defined variable-value pairs selectable with tab-completion of aliases from the command-line. The pEchoVar application provides a way of echoing an observed write to a variable with a new write with the same value to a different variable name. The uProcessWatch application is a way of monitoring the presence or absence of a set of MOOS processes and summarizing the collective status in a single MOOS variable. The uPokeDB application is 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.

This work is the product of a multi-year collaboration between the Center for Advanced System Technologies (CAST), Code 2501, of the Naval Undersea Warfare Center in Newport Rhode Island and the Department of Mechanical Engineering and the Computer Science and Artificial Intelligence Laboratory (CSAIL) at the Massachusetts Institute of Technology in Cambridge Massachusetts.

Points of contact for collaborators:

Dr. Michael R. Benjamin
Center for Advanced System Technologies
NUWC Division Newport Rhode Island
Michael.R.Benjamin@navy.mil
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

Sponsorship, and public release information:

This work was sponsored by Dr. Behzad Kamgar-Parsi and Dr. Don Wagner of the Office of Naval Research (ONR), Code 311. Information on Navy public release approval for this document can be obtained from the Technical Library at the Naval Undersea Warfare Center, Division Newport RI.

Contents

1 Overview

The MOOS-IvP autonomy tools described in this document are software applications that are typically running as part of an overall autonomy system running on a marine vehicle. They are each MOOS applications, meaning they are running and communicating with a MOOSDB application as depicted in Figure 1.

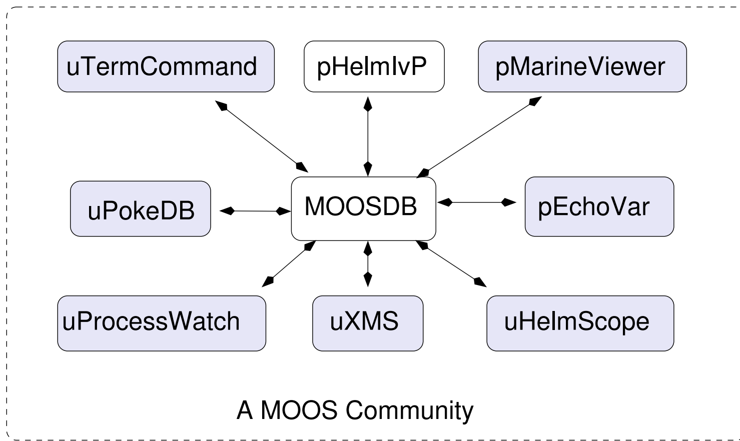


Figure 1: A MOOS community consists of a running MOOSDB application and a number of applications connected and communicating with each other via publish and subscribe interface. The pHelmIvP application provides the autonomy decision-making capapability and the other highlighted applications provide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

The scope of this paper is on these seven tools. 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.

Acronyms

MOOS stands for ”Mission Oriented Operating System” and its original use was for the Bluefin Oddysey III vehicle owned by MIT. IvP stands for ”Interval Programming” which is a mathematical programming model for multi-objective optimization. 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 to be distinct.

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 Paul 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 Paul Newman at Oxford and the most current version can be found at his website. The MOOS software available in the MOOS-IvP project is a subset 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 a different behavior coordination technique (multi-objective optimization) than the helm originally distributed with MOOS. The earlier helm (pHelm) is still available from the Oxford website.

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. MOOS is additionaly supported by sponsors in the U.K. 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.

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/NUWC includes additional MOOS utilities (seven of which are the topic of this document) and the IvP Helm and related behaviors. These modules are support on Linux and Mac OS X.

2 uHelmScope

2.1 Brief Overview

The uHelmScope application is a console based tool for monitoring output of the IvPHelm, 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  ============== (17)
 2   Helm Iteration: 66      (hz=0.38)(5)  (hz=0.35)(58)  (hz=0.56)(max)
 3   IvP functions:  1
 4   SolveTime:      0.00    (max=0.00)
 5   CreateTime:     0.02    (max=0.02)
 6   LoopTime:       0.02    (max=0.02)
 7   Halted:         false   (0 warnings)
 8 Helm Decision: [speed,0,4,21] [course,0,359,360]
 9   speed = 3.00
10   course = 177.00
11 Behaviors Active: ---------- (1)
12   waypt_survey (13.0) (pwt=100.00) (pcs=1227) (cpu=0.01) (upd=0/0)
13 Behaviors Running: --------- (0)
14 Behaviors Idle: ------------ (1)
15   waypt_return (22.8)
16 Behaviors Completed: ------- (0)
17
18 #  MOOSDB-SCOPE ------------------------------------ (Hit '#' to en/disable)
19 #
20 #  VarName           Source       Time     Community  VarValue
21 #  ----------------  -----------  -------  ---------  -----------
22 #  BHV_WARNING        n/a          n/a      n/a       n/a
23 #  AIS_REPORT_LOCAL  pTrans..rAIS 24.32    alpha      "NAME=alpha,TYPE=KAYAK,MOOSDB"+
24 #  DEPLOY*           iRemote      11.25    alpha      "true"
25 #  RETURN*           pHelmIvP     5.21     alpha      "false"
26
27 @  BEHAVIOR-POSTS TO MOOSDB ----------------------- (Hit '@' to en/disable)
28 @
29 @  MOOS Variable     Value
30 @  -------------     -------  (BEHAVIOR=waypt_survey)
31 @  PC_waypt_survey   -- ok --
32 @  WPT_STAT_LOCAL    vname=alpha,index=1,dist=80.47698,eta=26.83870
33 @  WPT_INDEX         1
34 @  VIEW_SEGLIST      label,alpha_waypt_survey : 30,-20:30,-100:90,-100: +
35 @  -------------     -------  (BEHAVIOR=waypt_return)
36 @  PC_waypt_return   RETURN = true
37 @  VIEW_SEGLIST      label,alpha_waypt_return : 0,0
48 @  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 1-16), a MOOSDB scope for a select subset of MOOS variables (lines 18-25), and a report on the MOOS variables published by the helm on the current iteration (lines 27-48). 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 1-16 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 4 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 5 is the total time needed by all active behaviors to produce their IvP function output. The loop time on line 6 is simply the sum of lines 4 and 5. The Boolean on line 7 is true only if the helm is halted on an emergency or critical error condition. Also on line 7 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 8, 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* behaviorsbut 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 12) 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 12 shows how many successful updates where made against how many attempts. A failed update attempt also generates a helm warning, counted on line 7. 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 36) 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 23 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 22 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 analyse 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.

```
KeyStroke  Function
---------  ---------------------------
   Spc     Pause and Update latest information once - now
   r/R     Resume information refresh
   h/H     Show this Help msg - 'r' to resume
```

```
    b/B     Toggle Show Idle/Completed Behavior Details
    t/T     Toggle truncation of column output
     m      Mask PWT_* UH_* STATE_* in Behavior-Posts Report
     M      Mask PC_* VIEW_* in Behavior-Posts Report
    s/S     Toggle Behavior State Vars in MOOSDB-Scope Report
    u/U     Unmask all variables in Behavior-Posts Report
    v/V     Toggle display of virgins in MOOSDB-Scope output
    [/]     Display Iteration 1 step prev/forward
    {/}     Display Iteration 10 steps prev/forward
    (/)     Display Iteration 100 steps prev/forward
     #      Toggle Show the MOOSDB-Scope Report
     @      Toggle Show the Behavior-Posts Report

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.

```
> uHelmScope -h
Usage: uHelmScope moosfile.moos [switches] [MOOSVARS]
  -t:  Column truncation is on (off by default)
  -c:  Exclude MOOS Vars in MOOS file from MOOSDB-Scope
  -x:  Suppress MOOSDB-Scope output block
  -p:  Suppress Behavior-Posts output block
  -v:  Suppress display of virgins in MOOSDB-Scope block
  -r:  Streaming (unpaused) output of helm iterations
  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 four 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_STATEVARS` and `IVPHELM_DOMAIN`. The first two are produced on each iteration of the helm, and the last two 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,
```

```
     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"
```

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.

Listing 4 - An example uHelmScope configuration block.

```
//------------------------------------------------------------
// uHelmScope configuration block

ProcessConfig = uHelmScope
{
  AppTick    = 1
  CommsTick  = 1

  PAUSED             = true     // All Parameters and Parameter-Values
  HZ_MEMORY          = 5, 100   // are __NOT__ Case Sensitive
  DISPLAY_MOOS_SCOPE = true
  DISPLAY_BHV_POSTS  = true
  DISPLAY_VIRGINS    = true
  DISPLAY_STATEVARS  = true
  TRUNCATED_OUTPUT   = false
  BEHAVIORS_CONCISE  = false

  VAR = BHV_WARNING, AIS_REPORT_LOCAL   // MOOS Variable names
}                                       // __ARE__ Case Sensitive
```

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 2.6. 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_IVP_DOMAIN: See Section 2.5.

3 pMarineViewer

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 *AIS reports* to the local community.

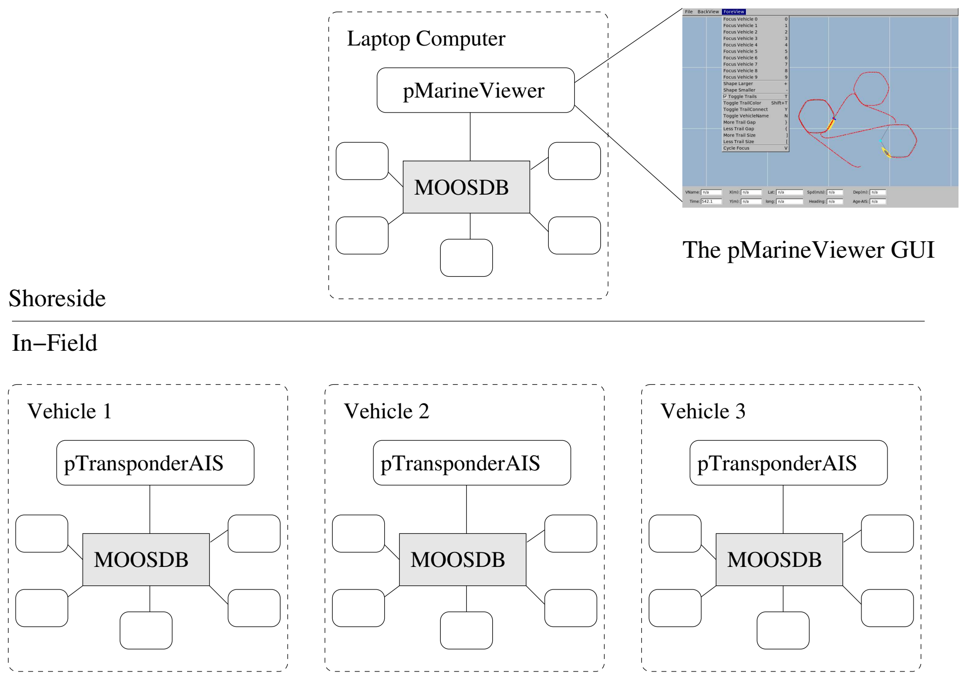


Figure 2: A common usage of the pMarineViewer is to have it running in a local MOOSDB community while receiving AIS reports on vehicle poise from other MOOS communities running on either real or simulated vehicles. The vehicles can also send messages with certain geometric information such as polygons 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. The key variable subscribed to by pMarineViewer is the variable AIS_REPORT, which has the following structure given by an example:

```
AIS_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"
```

Reports from different vehicles are sorted by their vehicle name and stored in histories locally in the pMarineViewer application. The AIS_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.

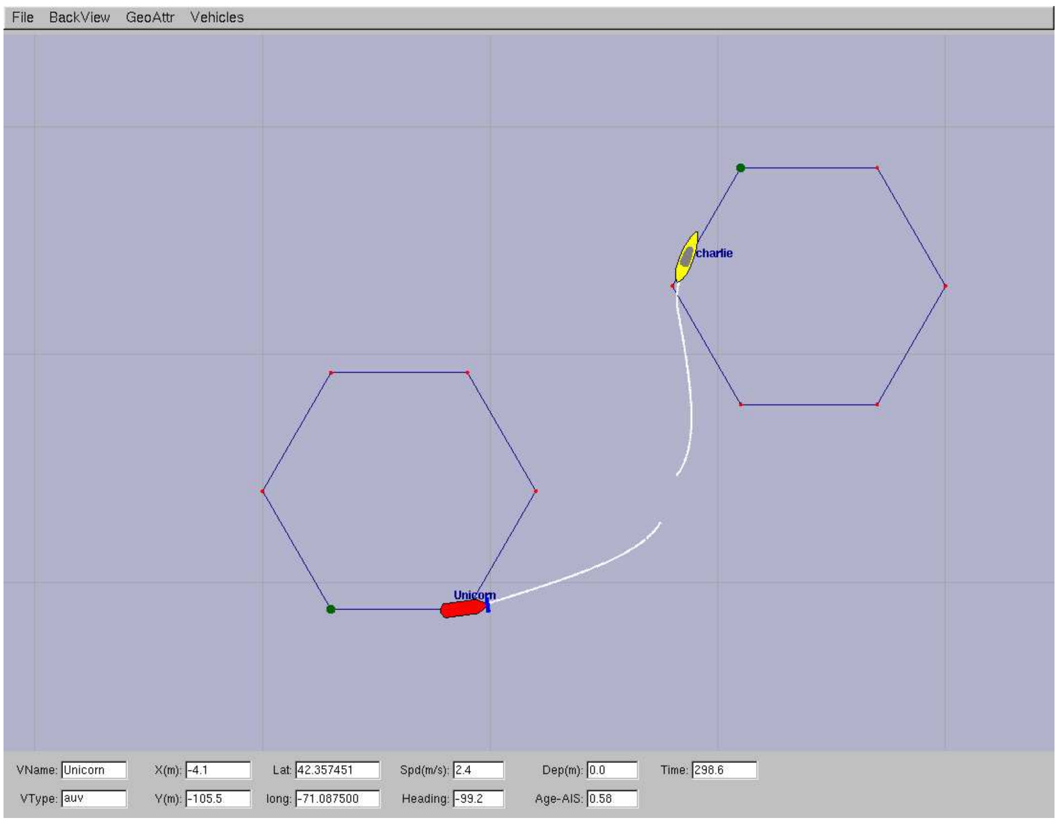


Figure 3: A screen shot of the pMarineViewer application running with two vehicles - one kayak platform, and one AUV platform. The unicorn AUV platform is the *active* platform meaning the data fields on the bottom reflect the data for this platform.

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 in Listing 5.

Listing 5 - Description of the on-screen fields of pMarineViewer.

```
Field    Description
-----    -----------
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 ($0-359.9$).

Depth    The depth of the active vehicle given in meters.

Age-AIS  The elapsed time in seconds since the last received AIS report for
         the active vehicle.
Time     Time in seconds since the pMarineViewer process launched.

```

In simulation, the age of the AIS report is likely to remain zero as shown in the figure, but when operating on the water, monitoring the AIS 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.

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'.

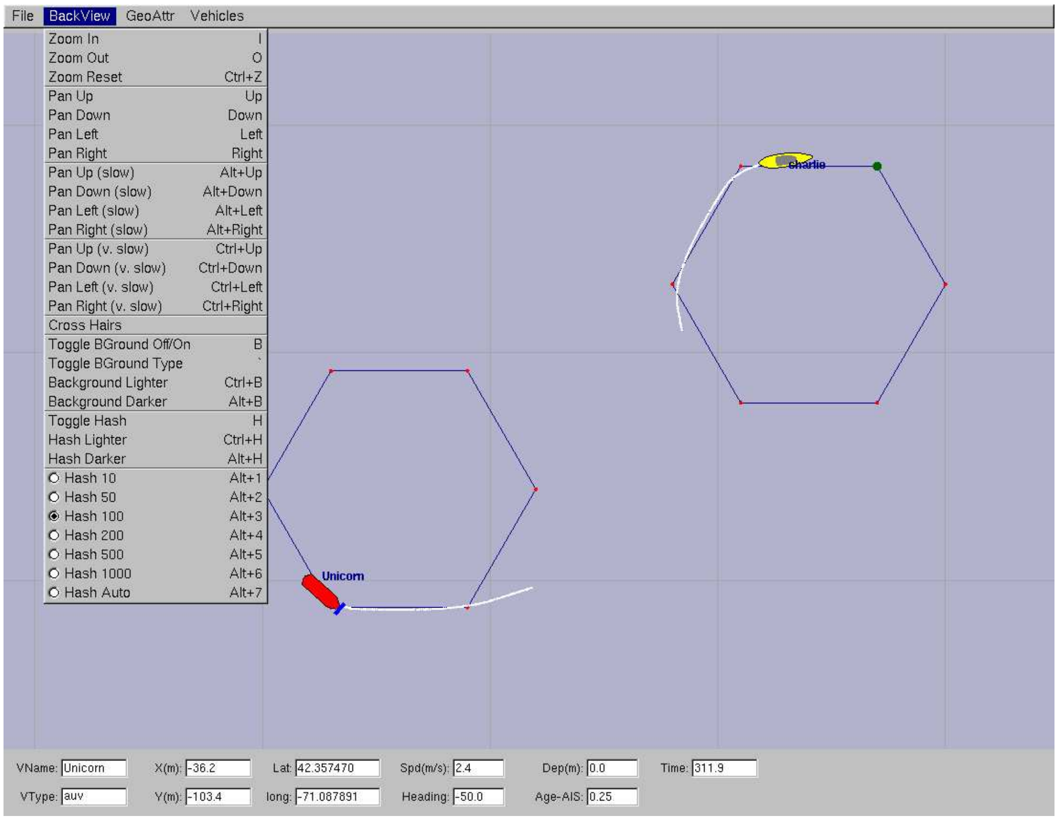


Figure 4: The *BackView* pull-down menu of the pMarineViewer lists the options, with hot-keys, for affecting rendering aspects of the geo-display background.

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 3.6. 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 positive integer not greater than 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 hexagons in Figure 5 for example represents the set of waypoints being used by the vehicles shown.

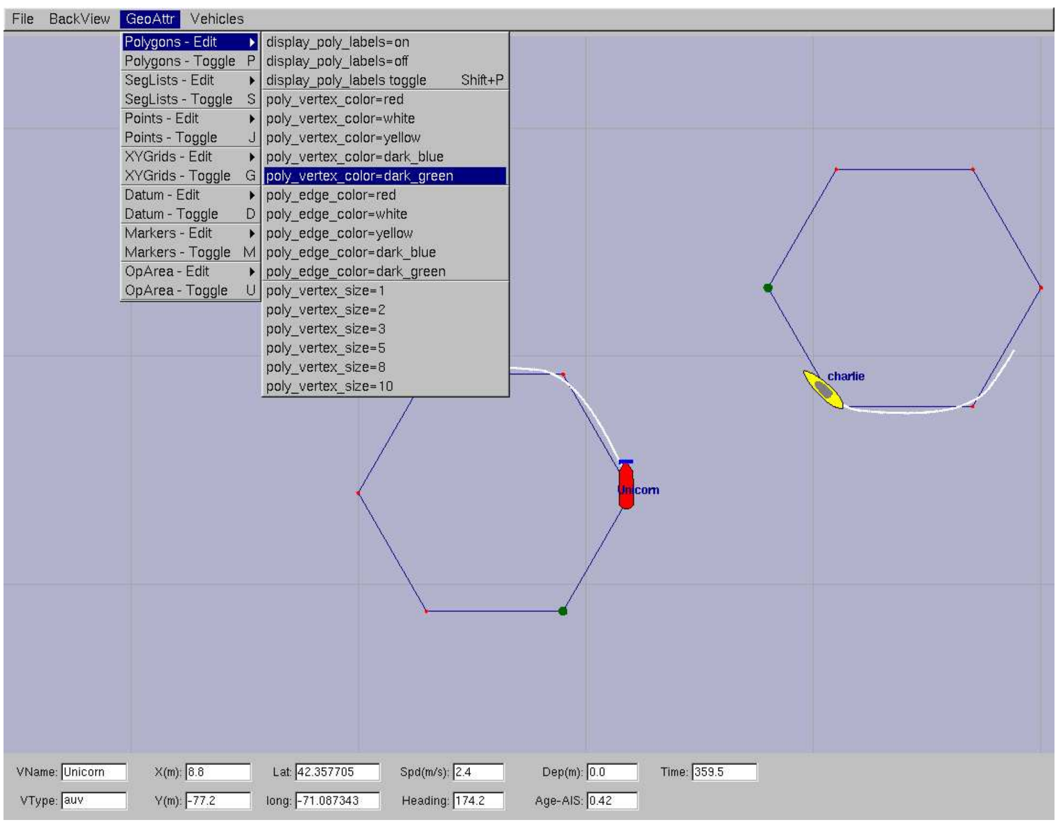


Figure 5: The *GeoAttributes* pull-down menu of the pMarineViewer lists the options and hot keys for affecting the rendering of geometric objects

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 AIS reports. If an AIS report arrives for a previously unknown vehicle, it is assigned a new index.

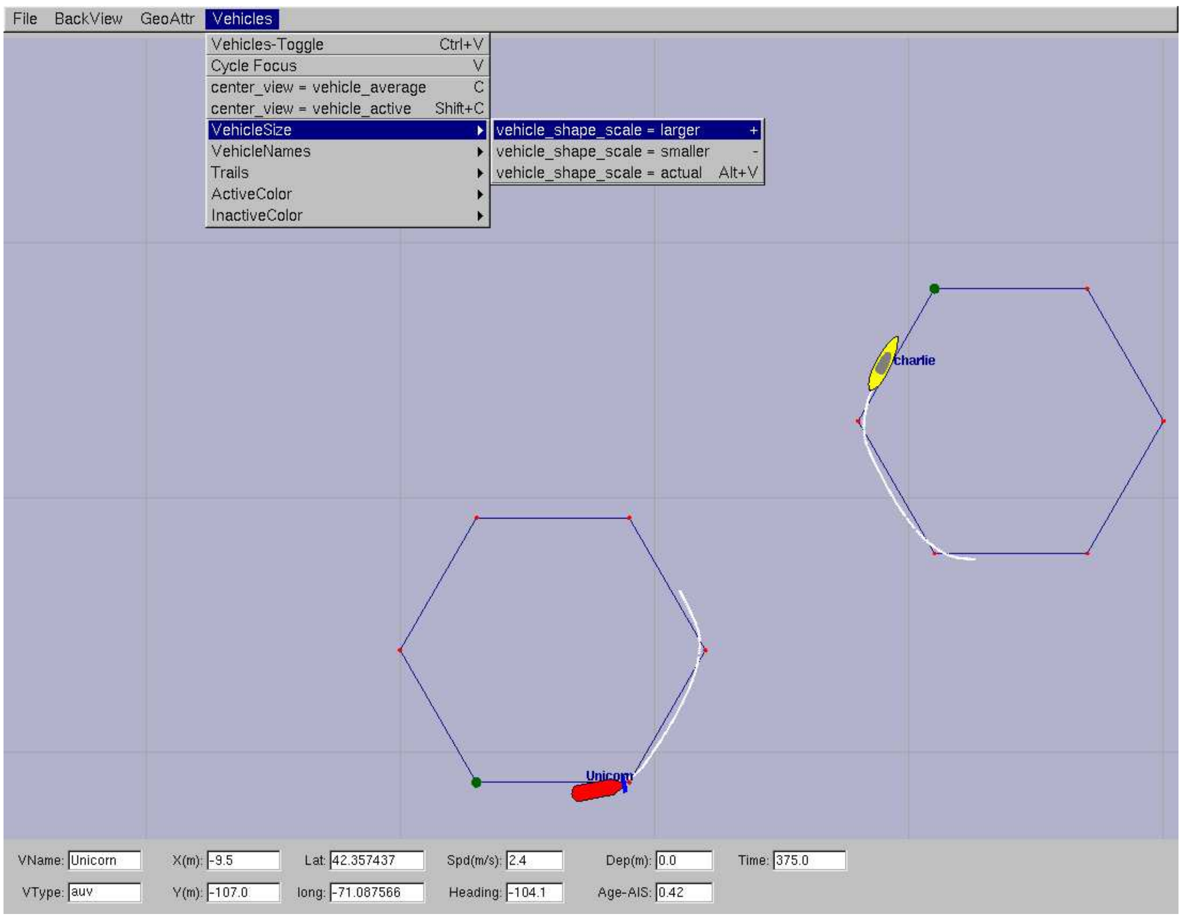


Figure 6: The *ForeView* pull-down menu of the pMarineViewer lists the options, with hot-keys, for affecting rendering aspects of the objects on the geo-display foreground, such as vehicles and vehicle track history.

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 AIS 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.

3.4 Displayable Vehicle Shapes, Markers and other Geometric Objects

3.4.1 Displayable Vehicle Shapes

The shape rendered for a particular vehicle depends on the *type* of vehicle indicated in the AIS 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 7.

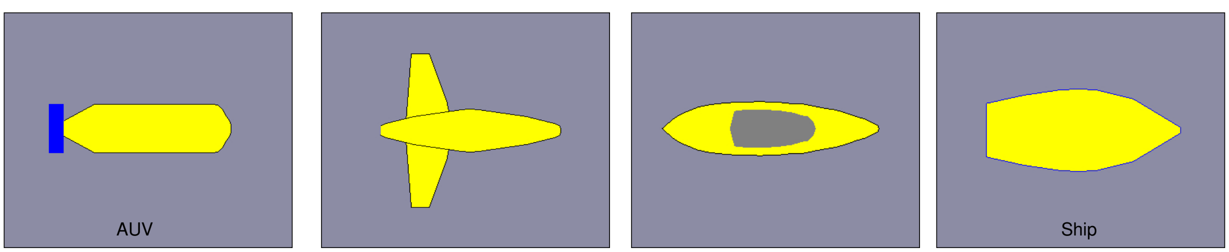


Figure 7: Vehicle types known to the pMarineViewer.

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.

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 five types of markers are shown in Figure 8. They are configured in the pMarineViewer configuration block of the MOOS file with the following format:

```
marker  = type=efield,x=100,y=20,SCALE=4.3,label=alpha,COLOR=red     // parameters and types
marker  = type=square,lat=42.358,lon=-71.0874,scale=2,color=blue     // are case insensitive
```

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 *scale* parameter is by default 1 and simply scales linearly the size of the shape. 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.

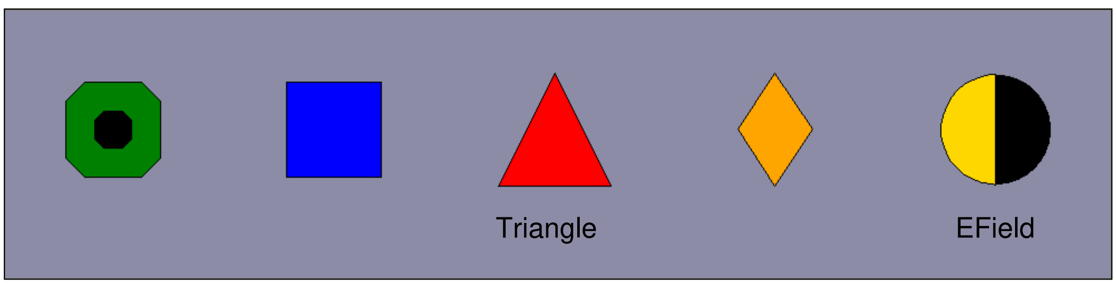


Figure 8: Marker types known to the pMarineViewer.

The color parameter is optional and markers have the default colors shown in Figure 8. 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.

3.4.3 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 AIS_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,nyak201-LOITER:85,-9:100,-35:85,-61:55,-61:40,-35:55,-9"
VIEW_POINT   = 10.00,-80.00,5,nyak200
VIEW_SEGLIST = "label,nyak201-WAYPOINT: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 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 6.

Listing 6 - An example pMarineViewer configuration block.

```
LatOrigin  =   47.7319
LongOrigin = -122.8500

//-----------------------------------------------
// pMarineViewer configuration  block

ProcessConfig = pMarineViewer
{
  // Standard MOOS parameters affecting comms and execution
  AppTick    = 4
  CommsTick  = 4

  // Parameters and their default values
  HASH_VIEW     = false
  HASH_DELTA    = 50
  HASH_SHADE    = 0.65
  BACK_SHADE    = 0.70
  TRAIL_VIEW    = true
  TRAIL_SIZE    = 0.1
  TRAIL_GAP     = 1.0
  TIFF_VIEW     = true
  ZOOM          = 1.0
  DISPLAY_VNAME = false
```

```
24    VERBOSE       = false
25
26    // Setting the vehicle colors - default is yellow
27    VEHICOLOR     = nyak200,dark_blue
28    VEHICOLOR     = nyak201,0.0,0.0,0.545
29    VEHICOLOR     = nyak202,hex:00,00,8b
30
31    // All polygon parameters are optional - defaults are shown
32    // They can also be set dynamically in the GUI in the GeoAttrs pull-down menu
33    polygon_edge_color   = yellow
34    polygon_vertex_color = red
35    polygon_label_color  = khaki
36    polygon_edge_width   = 1.0
37    polygon_vertex_size  = 3.0
38    polygon_viewable_all = true;
39    polygon_viewable_labels = true;
40
41    // All seglist parameters are optional - defaults are shown
42    // They can also be set dynamically in the GUI in the GeoAttrs pull-down menu
43    seglist_edge_color   = white
44    seglist_vertex_color = dark_blue
45    seglist_label_color  = orange
46    seglist_edge_width   = 1.0
47    seglist_vertex_size  = 3.0
48    seglist_viewable_all = true;
49    seglist_viewable_labels = true;
50
51    // All point parameters are optional - defaults are shown
52    // They can also be set dynamically in the GUI in the GeoAttrs pull-down menu
53    point_vertex_size  = 4.0;
54    point_vertex_color = yellow
55    point_viewable_all = true;
56    point_viewable_labels = true;
57  }
```

Color references as in lines 27-29 can be made by name or by hexadecimal or decimal notation. (All three colors in lines 27-29 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 24 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 `AIS_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 7.

Listing 7 - An example pMarineViewer console output.

```
// Example pMarineViewer console output NOT in verbose mode

13.56 > ****..
13.82 > **..
14.08 > **..
14.35 > **..
```

```
14.61 > ****.P.P
14.88 > **..
15.14 > **..

// Example pMarineViewer console output in verbose mode

15.42 >
   AIS(nyak201)
   AIS(nyak200)
   Point(nyak201_wpt)
   Point(nyak200_wpt)

15.59 >
   Point(nyak201)
   Poly(nyak201-LOITER)
   AIS(nyak201)
   AIS(nyak200)
   Point(nyak200)
   Poly(nyak200-LOITER)
```

3.6 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 8.

Listing 8 - An example .info file for the pMarineViewer

```
// Lines may be in any order, blank lines are ok
// Comments begin with double slashes

datum_lat  = 47.731900
datum_lon  = -122.85000
lat_north  = 47.768868
lat_south  = 47.709761
lon_west   = -122.882080
lon_east   = -122.794189
```

All six parameters are mandatory. The two datum lines indicate where $(0, 0)$ in local coordinates is in earth coordinates. The `lat_north` parameters correlates 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 9 with two images of Dabob Bay in Washington State. Both image files where created from resources at www.maps.google.com.

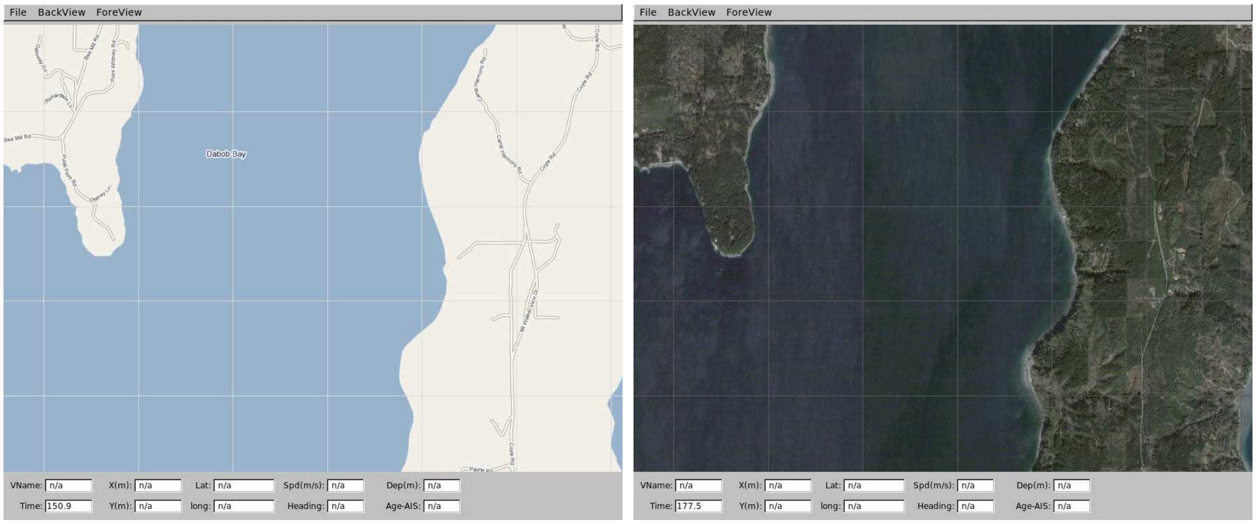


Figure 9: Two images loaded for use in the geo display mode of pMarineViewer. The user can toggle between both as desired during operation.

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.

3.7 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., buttons to re-deploy or manipulate vehicle missions, can be easily 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 the uTermCommand tool described in Section 5 provides one way to facilitate this. But 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.

3.8 Publications and Subscriptions for pMarineViewer

Variables published by the pMarineViewer application

- `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 3.7. 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.
- `VIEWER_CONNECTED`: This variable is published once when the viewer connects to the MOOSDB. It can be used by other applications to prompt them to re-publish information that the viewer may need. It is used in the pHelmIvP application for example, to clear a local buffer used to prevent successive identical publications to its variables.

Variables subscribed for by pMarineViewer application

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

  ```
  AIS_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"
  ```

- `AIS_REPORT_LOCAL`: This serves the same purpose as the above variable. In some simulation cases this variable is used.
- `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.
- `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.
- `GRID_CONFIG`: A string representation of a grid. This initializes and registers a new grid with the viewer.
- `GRID_DELTA`: A string representation of a change in values for a given grid and specific grid cells with new value for each given cell.

4 uXMS

4.1 Brief Overview

The uXMS application is a terminal based tool for live scoping on a MOOSDB process. Since it is not dependent on any graphic libraries it is more likely to run out-of-the-box on machines that may not have proper 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. It can be run in a mode where screen updates only occur at the user's request. For these reasons, it is a good choice when bandwidth is an issue. It is also possible to have multiple versions of uXMS connected to the same MOOSDB. The uXMS tool was meant to be an alternative to the more feature-rich, high-bandwidth, GUI-based uMS process written and distributed on the Oxford MOOS website.

4.2 Configuration Parameters for uXMS

Configuration for uXMS amounts to specifying the *scope list* - those variables uXMS will register with the MOOSDB for updates and display in the console output. The scope list can be augmented in two ways - from the command line, and from the uXMS configuration block of a MOOS file. In the MOOS file, lines have the form:

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

Duplicates, should they occur, are simply ignored. An example configuration is given in Listing 9.

Listing 9 - An example uXMS configuration block.

```
//------------------------------------------------------------------
// uXMS configuration block

ProcessConfig = uXMS
{
  AppTick    = 5
  CommsTick  = 5

  // Navigation information (Or use -nav on the command line)
  VAR  = NAV_X, NAV_Y, NAV_HEADING, NAV_SPEED, NAV_DEPTH

  // Helm output information (Or use -helm on the command line)
  VAR  = DESIRED_HEADING, DESIRED_SPEED, DESIRED_DEPTH

  // More helm information (Or use -helm on the command line)
  VAR  = BHV_WARNING, BHV_ERROR, MOOS_MANUAL_OVERIDE
  VAR  = HELM_IPF_COUNT, HELM_ACTIVE_BHV, HELM_NONIDLE_BHV
  VAR  = DEPLOY, RETURN, STATION_KEEP

  // PID output information (or use -pid on the command line)
  VAR  = DESIRED_RUDDER, DESIRED_THRUST, DESIRED_ELEVATOR

  // uProcessWatch output (or use -proc on the command line)
  VAR  = PROC_WATCH_SUMMARY, PROC_WATCH_EVENT

  // Display parameters - Values shown are defaults
  PAUSED                = true    // false if -r on cmd-line
```

```
  DISPLAY_VIRGINS       = true    // false if -v on cmd-line
  DISPLAY_EMPTY_STRINGS = true    // false if -e on cmd-line
  DISPLAY_SOURCE        = false   // true  if -s on cmd-line
  DISPLAY_TIME          = false   // true  if -t on cmd-line
  DISPLAY_COMMUNITY     = false   // true  if -c on cmd-line
}
```

4.3 Command Line Arguments 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:

```
> uXMS -h
Usage: uXMS [file.moos] [-nav] [-helm] [-pid] [-proc]
       [--clean|-c] [--resumed|-r] [--virgins|-v]
       [--source|-s] [--time|-t] [--all|-a]
       [server_host=value] [server_port=value]
       [MOOS_VARIABLES]

[file.moos] Filename to get configuration parameters
[-nav]      Auto-subscribe for NAV_* variables
[-helm]     Auto-subscribe for IvPHelm variables
[-pid]      Auto-subscribe for PID (DESIRED_*) vars
[-proc]     Auto-subscribe for uProcessWatch vars
[-c]        Ignore scope variables in file.moos
[-r]        Start in data-streaming mode
[-v]        Don't display virgin variables
[-s]        Show the Source field in the data report
[-t]        Show the Time field in the data report
[-a]        Show ALL MOOS variables in the MOOSDB
[server_host=value]  Connect to MOOSDB at IP=value
[server_port=value]  Connect to MOOSDB at port=value
```

The -nav, -pid, -helm, and -proc switches are convenient aliases for common groups of variables. See Listing 9. 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 -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"
```

4.4 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 10. 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 10 - The uXMS console output with -proc, -nav and -pid command line options.

```
 1    VarName               (S)  (T)  (C) VarValue
 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 unexpanded. Listing 11 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 9. 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 11 - The help-menu on the uXMS console.

```
 KeyStroke    Function
 ---------    ---------------------------
     s        Surpress Source of variables
     S        Display Source of variables
     t        Surpress Time of variables
     T        Display Time of variables
     c        Surpress Community of variables
     C        Display Community of variables
     v        Surpress virgin variables
     V        Display virgin variables
     e        Surpress empty strings
     E        Display empty strings
Space/u/U     Update information once - now
    p/P       Pause information refresh
    r/R       resume information refresh
    h/H       Show this Help msg - 'R' to resume
```

The three middle columns can be expanded as shown in Listing 12. 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 12 - An example uXMS console output with all fields expanded.

```
VarName               (S)ource      (T)ime      (C)ommunity   VarValue
----------------      ----------    ----------  ----------     ----------- (4)
PROC_WATCH_SUMMARY     uProcessWatch 364.486     nyak200        "AWOL: pEchoVar"
NAV_X                  pEchoVar      365.512     nyak200        10
NAV_Y                  pEchoVar      365.512     nyak200        -10
NAV_HEADING            pEchoVar      365.512     nyak200        180
NAV_SPEED              pEchoVar      365.512     nyak200        0
DESIRED_RUDDER         pMarinePID    365.512     nyak200        0
DESIRED_THRUST         pMarinePID    365.512     nyak200        0
```

Variables that have yet to be written, as lines 8 and 11 in Listing 10, 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.

4.5 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.

4.6 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.

4.7 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 4.2.

5 uTermCommand

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 13.

Listing 13 - 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 13, 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 14 shows what the console output would look like given the configuration parameters of Listing 13. 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 14 - Console output at start-up.

```
  Cue                      VarName                         VarValue
  -----------             ---------------                  --------------
  deploy_true              DEPLOY                          "true"
  deploy_false             DEPLOY                          "false"
  return_true              RETURN                          "true"
  return_false             RETURN                          "false"
  station_true             STATION_KEEP                    "true"
  station_false            STATION_KEEP                    "false"

>
```

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 15. 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 15 - Console output after typing a single character 'r'.

```
  Cue                      VarName                         VarValue
  -----------             ---------------                  --------------
  return_true              RETURN                          "true"
  return_false             RETURN                          "false"

> 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 16.

Listing 16 - Console output when a single command is identified.

```
  Cue                      VarName                         VarValue
  -----------             ---------------                  --------------
  return_true              RETURN                          "true" <-- SELECT

> 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 10.

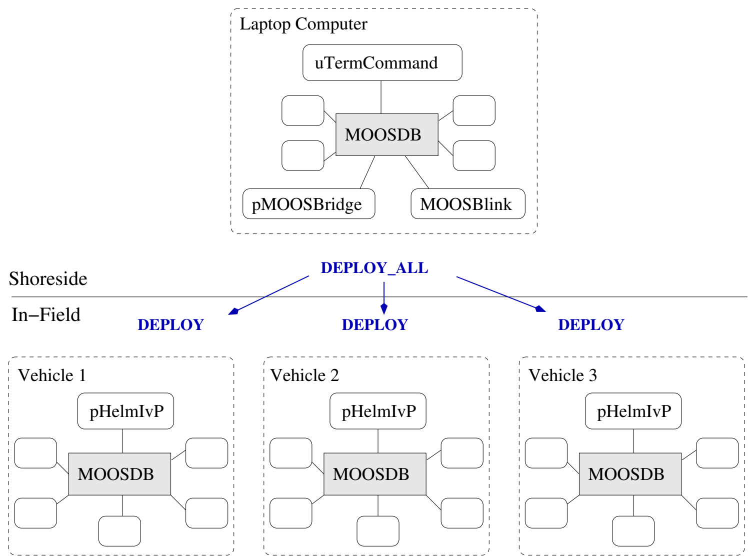


Figure 10: A common usage of the uTermCommand application for command and control.

In the example below in Listing 17, 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 17 - 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
```

```
  CMD  = 200_deploy_false   -->  DEPLOY_200        --> false
  CMD  = 200_return_true    -->  RETURN_200        --> true
  CMD  = 200_return_false   -->  RETURN_200        --> false

  CMD  = 201_deploy_true    -->  DEPLOY_201        --> true
  CMD  = 201_deploy_false   -->  DEPLOY_201        --> false
  CMD  = 201_return_true    -->  RETURN_201        --> true
  CMD  = 201_return_false   -->  RETURN_201        --> false
 }
```

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 18. 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 18 - 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 Publications and Subscriptions for uTermCommand

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.

Variables subscribed for by the uPokeDB application

- `None`

6 pEchoVar

6.1 Brief Overview

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.

6.2 Configuration Parameters for pEchoVar

The echo configurations are set in the pEchoVar configuration block of the MOOS file. Each mapping requires one line of the form:

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

The *source_variable* and *target_variable* fields 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. An example configuration is given in Listing 19.

Listing 19 - 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 }
```

6.3 Configuring for Vehicle Simulation with pEchoVar

When in simulation mode with iMarineSim, 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 19. The simulator instead produces `MARINESIM_*` values which can be echoed as `NAV_*` values as shown in Listing 20.

Listing 20 - An example pEchoVar configuration block during simulation.

```
//------------------------------------------------------------
// pEchoVar configuration block (for simulation mode)

ProcessConfig = pEchoVar
```

```
{
  AppTick   = 20
  CommsTick = 20

  Echo = MARINESIM_X       ->  NAV_X
  Echo = MARINESIM_Y       ->  NAV_Y
  Echo = MARINESIM_HEADING ->  NAV_HEADING
  Echo = MARINESIM_SPEED   ->  NAV_SPEED
}
```

6.4 Publications and Subscriptions for pEchoVar

Variables published by the pEchoVar application

- `USER-DEFINED`: Each echo mapping described in Section 6.2 has a variable being echoed from, and a variable being echoed to. The variable being echoed *to* is published by `pEchoVar`.

Variables subscribed for by the pEchoVar application

- `USER-DEFINED`: Each echo mapping described in Section 6.2 has a variable being echoed from, and a variable being echoed to. The variable being echoed *from* is subscribed for by `pEchoVar`.

7 uProcessWatch

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 10, 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 12. 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 21.

Listing 21 - An example uProcessWatch configuration block.

```
//------------------------------------------------------------
// uProcessWatch configuration block

ProcessConfig = uProcessWatch
{
  AppTick  = 2
  CommsTick = 2

  // Declare the watch-list below
  WATCH  = pHelmIvP
  WATCH  = iMarineSim
  WATCH  = pEchoVar
  WATCH  = pLogger
  WATCH  = pMarinePID
  WATCH  = pTransponderAIS
  WATCH  = pMOOSBridge*
}
```

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 21 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 10, or composed of a comma-separated list of missing processes identified as being AWOL (absent without leave), as shown in line 3 of Listing 12.

- `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

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. Unlike iRemote it associates a variable-value pair with a key *word* rather than a keyboard key. 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 post-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’’ doublestr:=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.

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"
```

8.3 Session Output from uPokeDB

The output in Listing 22 shows an example 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 22 - 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
```

```
  ----------------      ----------  ----------   -------------
  DEPLOY                 pHelmIvP    1.87        "false"
  RETURN                 pHelmIvP    1.87        "false"

AFTER Poking the MOOSDB
  VarName               (S)ource    (T)ime      VarValue
  ----------------      ----------  ----------   -------------
  DEPLOY                 uPokeDB     8.48        "true"
  RETURN                 uPokeDB     8.48        "true"
```

8.4 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 8.3, it subscribes for the same variables it is asked to poke, so it can generate its before-and-after reports.

9 Appendix 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".

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)

Index