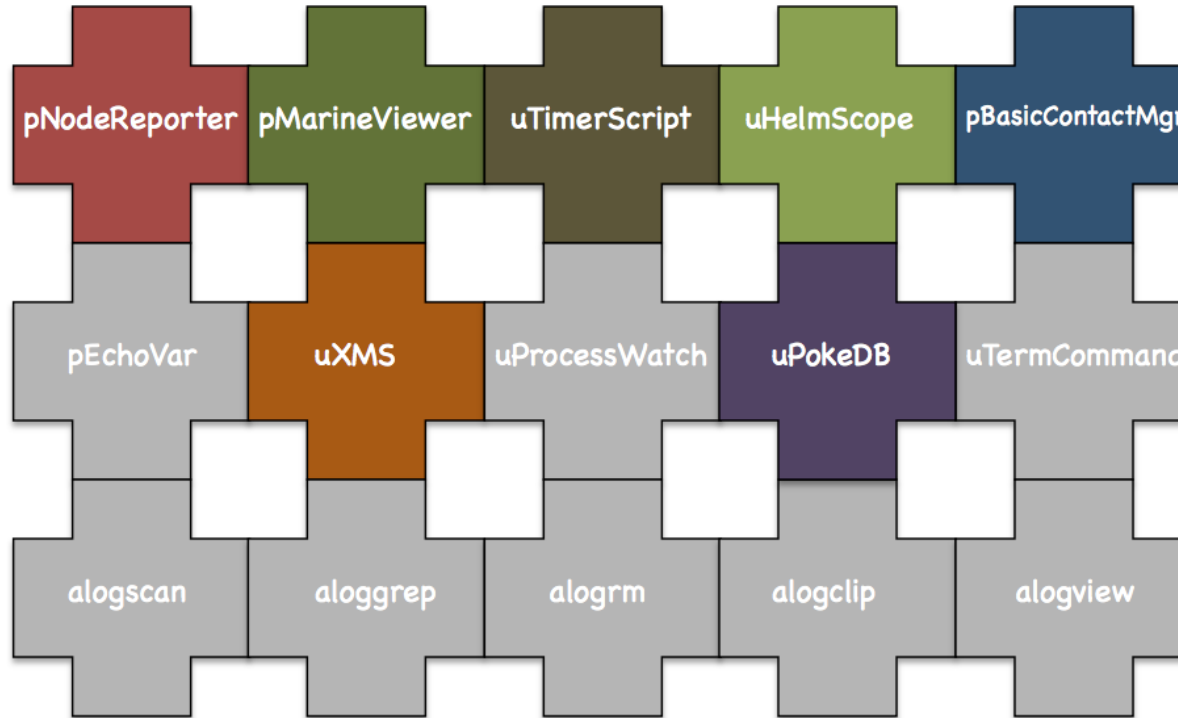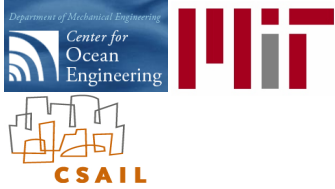# MOOS-IvP Autonomy Tools
(A Mini Tutorial)
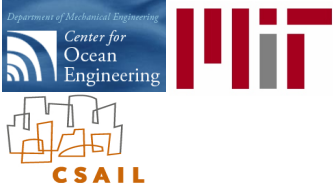
Michael Benjamin
MIT Department of Mechanical Engineering
Computer Science and Artificial Intelligence Laboratory

Acknowledgments

Mini-Tutorial Objectives and Structure

Objectives:

Introduce with some depth and examples, the following tools (software applications):

- pMarineViewer -   A GUI tool for rendering vehicle operations onto an geo-referenced display.
- pNodeReporter -   Captures vehicle state information and publishes a summary string
- uPokeDB -   A tool for poking the MOOSDB from the command line.
- uXMS -   A tool for focused scoping of the MOOSDB from the console
- uHelmScope -   A specialized scope on IvP Helm status and recent history
- pBasicContactMgr -   A simple manager of vehicle contacts, and generation of alerts
- uTimerScript -   A tool for scripting (possibly conditional and random) pokes to the MOOSDB

Structure:

Example missions will be used to demonstrate each tool.

- These example missions are available along with the MOOS-IvP source code at www.moos-ivp.org
- The example missions can be found under moos-ivp/ivp/missions/.
- It is recommended that tutorial participants download and be able to run these on their laptop computers.

Not All Tools Are Created Equal
Relative Tool Size by Line Count
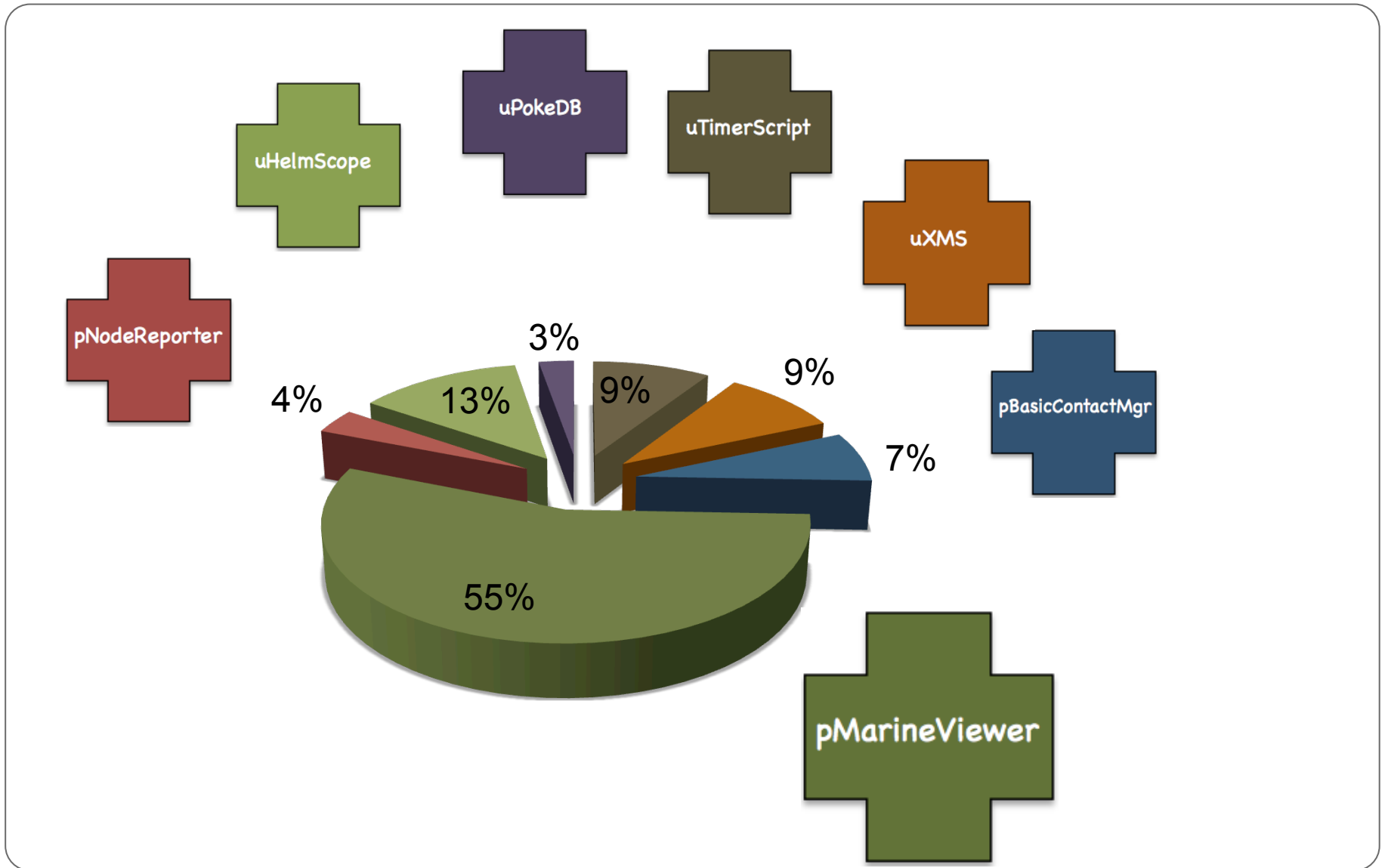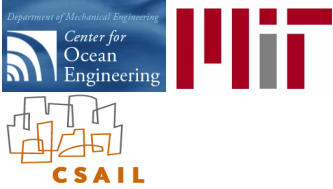
MOOS-IvP Autonomy Tools

Q: What is an Autonomy Tool?

Ans: A software module/application that supports either (a) the on-board autonomy, (b) pre-mission planning, (c) topside mission monitoring/control, (d) post-mission analysis.

Q: What is the relationship between an Autonomy Tool and the IvP Helm?

Ans: The IvP Helm is a decision engine that drives the vehicle with heading/speed/depth commands.

- Some tools are specific to the IvP Helm.
- Some tools are general (helm agnostic).
- Some are general but do have some IvP Helm hooks

Q: Where can the Autonomy Tools and documentation be found?

Ans: www.moos-ivp.org

A Walk Through the Alpha Mission (1)

(1) The "alpha" vehicle

(2) When the simulation is first launched, the vehicle is in "manual override" mode and the helm is DISENGAGED
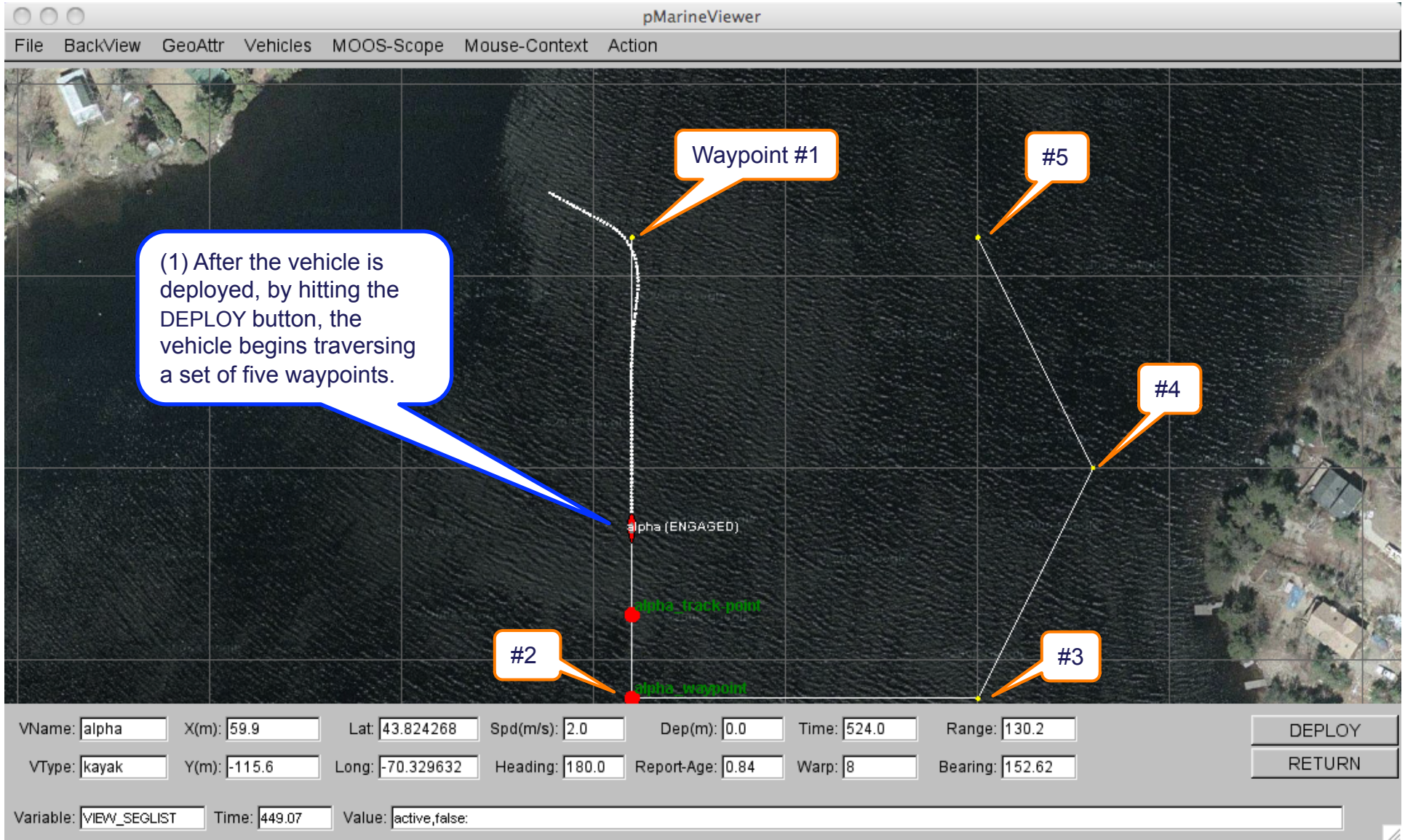
(3) Your window into the simulation is via the pMarineViewer application (one of the tools in this tutorial).

(4) The vehicle may be deployed by hitting the DEPLOY button. This simply asks the pMarineViewer application to "pokes" the MOOSDB with DEPLOY=true.

A Walk Through the Alpha Mission (2)

A Walk Through the Alpha Mission (3)

A Walk Through the Alpha Mission (4)

A Walk Through the Alpha Mission (5)

The uXMS Utility:
Scoping the MOOSDB from the Console
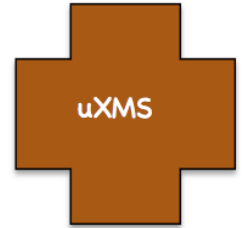
MOOS Modules:

- uXMS - A tool for **focused** scoping of the MOOSDB from the **console**.

- uPokeDB - A tool for poking the MOOSDB from the command line.

- pMarineViewer - A GUI tool for rendering vehicle operations onto an geo-referenced display.

- pNodeReporter - Captures vehicle state information and publishes a summary string.

- uHelmScope - A specialized scope on IvP Helm status and recent history.

- pBasicContactMgr - A simple manager of vehicle contacts, and generation of alerts.

- uTimerScript - A tool for scripting (possibly conditional and random) pokes to the MOOSDB.
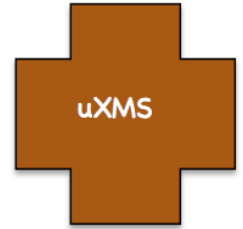
The uXMS Utility:
Definition of a MOOS Scope and MOOS Community

uXMS

What is a MOOS *scope*?

A scope is at tool for monitoring the current state of variables published in a MOOS community.
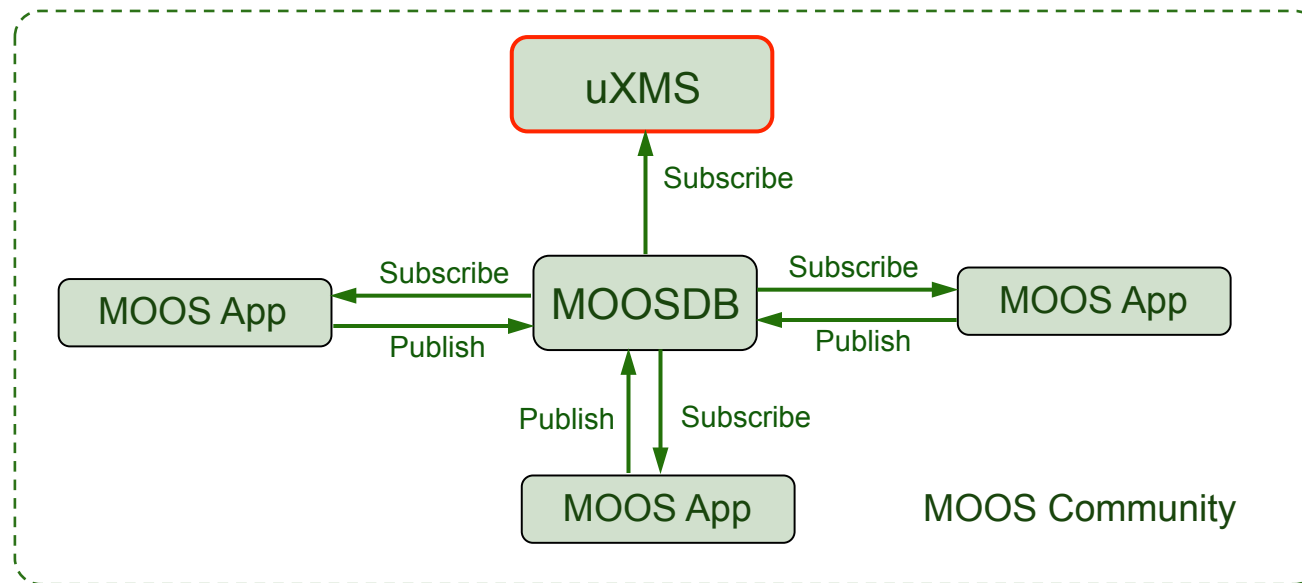
What is a MOOS *community*?
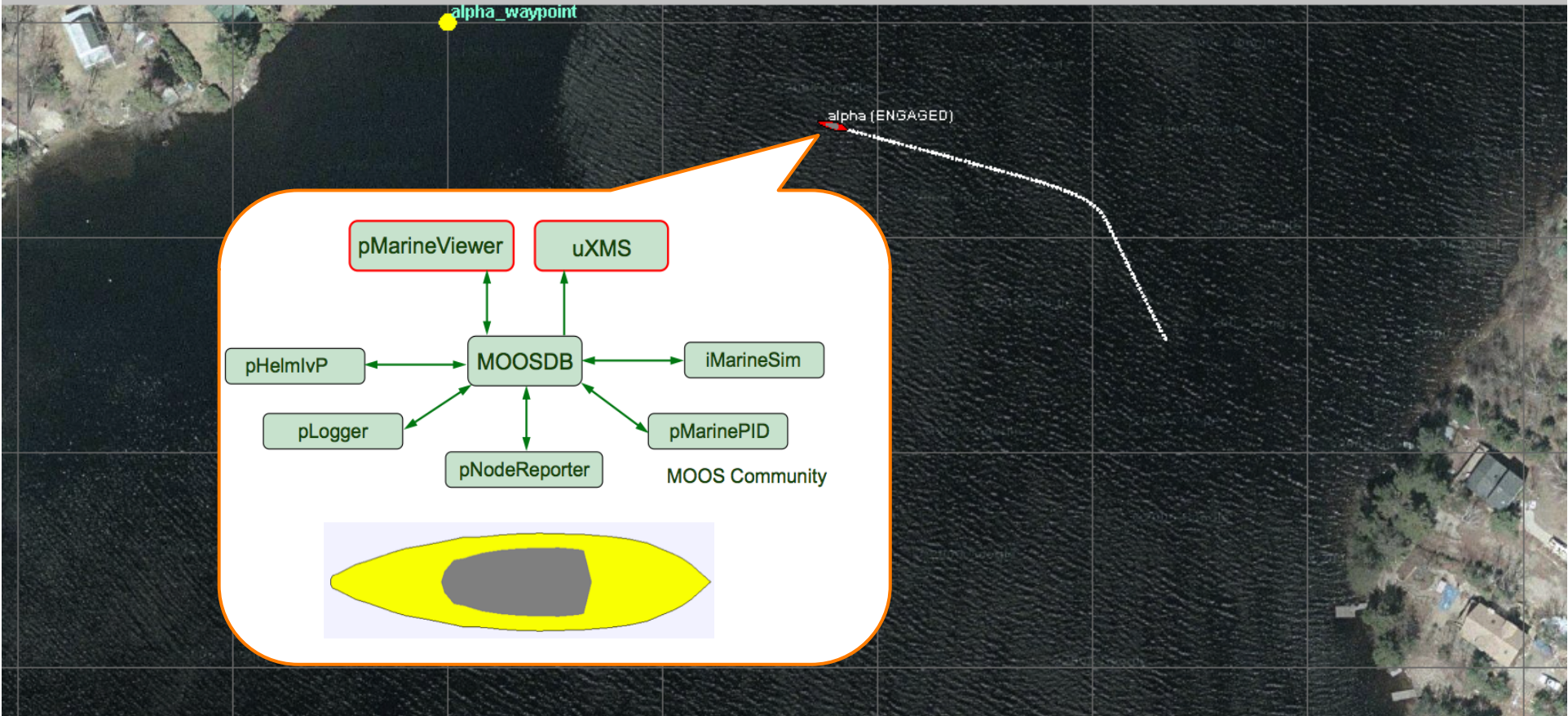
- A collection of MOOS applications connected to the single MOOSDB application.
- Each application interface is defined by what variables it publishes and subscribes to.
- The MOOSDB contains a snapshot of all the current variables – their values and other info.
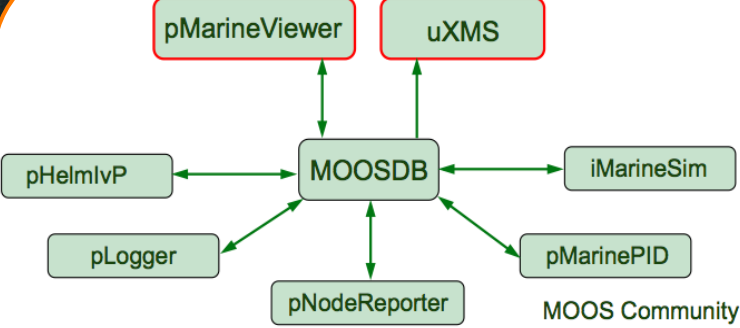
The uXMS Utility:
The MOOS Community in the Alpha Mission
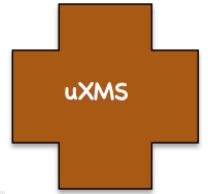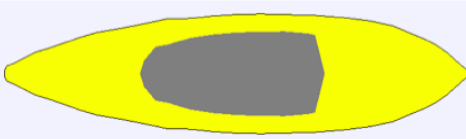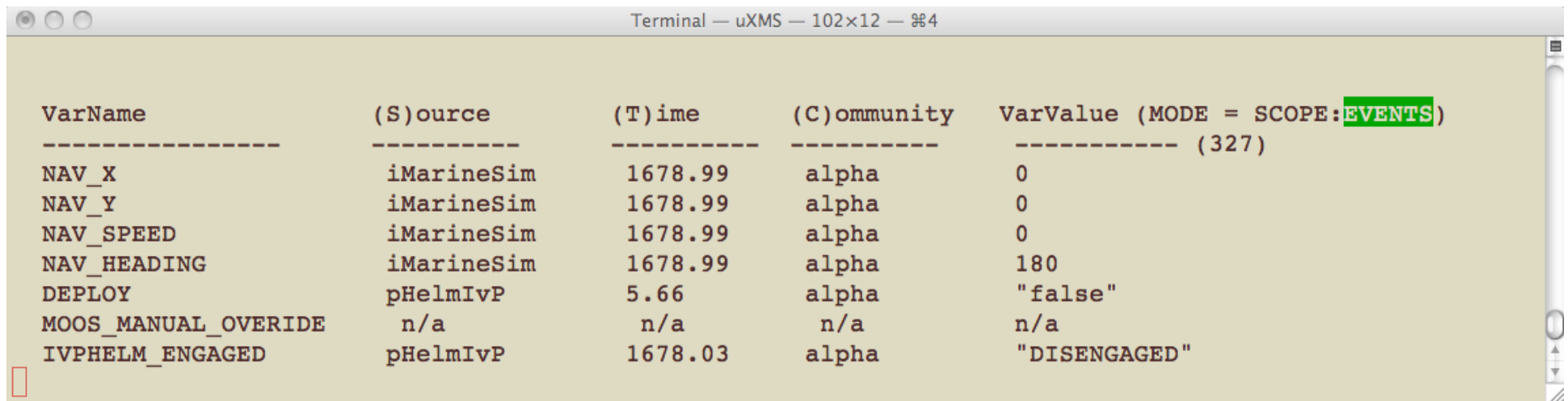
The uXMS Utility:
Launching from the command line
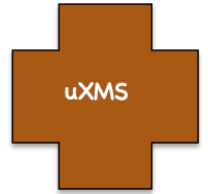
The uXMS utility is launched from the command line:

```
$ uXMS alpha.moos NAV_X NAV_Y NAV_SPEED NAV_HEADING DEPLOY
  MOOS_MANUAL_OVERIDE DEPLOY IVPHELM_ENGAGED
```

Upon launching, reports are written to the console:
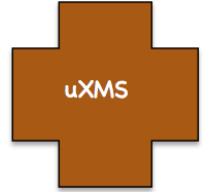
```
Terminal — uXMS — 102×12 — ⌘4

VarName                 (S)ource        (T)ime      (C)ommunity     VarValue (MODE = SCOPE:EVENTS)
----------------        ----------      ----------  ----------      ------------ (327)
NAV_X                   iMarineSim      1678.99     alpha           0
NAV_Y                   iMarineSim      1678.99     alpha           0
NAV_SPEED               iMarineSim      1678.99     alpha           0
NAV_HEADING             iMarineSim      1678.99     alpha           180
DEPLOY                  pHelmIvP        5.66        alpha           "false"
MOOS_MANUAL_OVERIDE      n/a             n/a         n/a            n/a
IVPHELM_ENGAGED         pHelmIvP        1678.03     alpha           "DISENGAGED"
```

uXMS operates by simply writing a "report" to the console on each iteration.
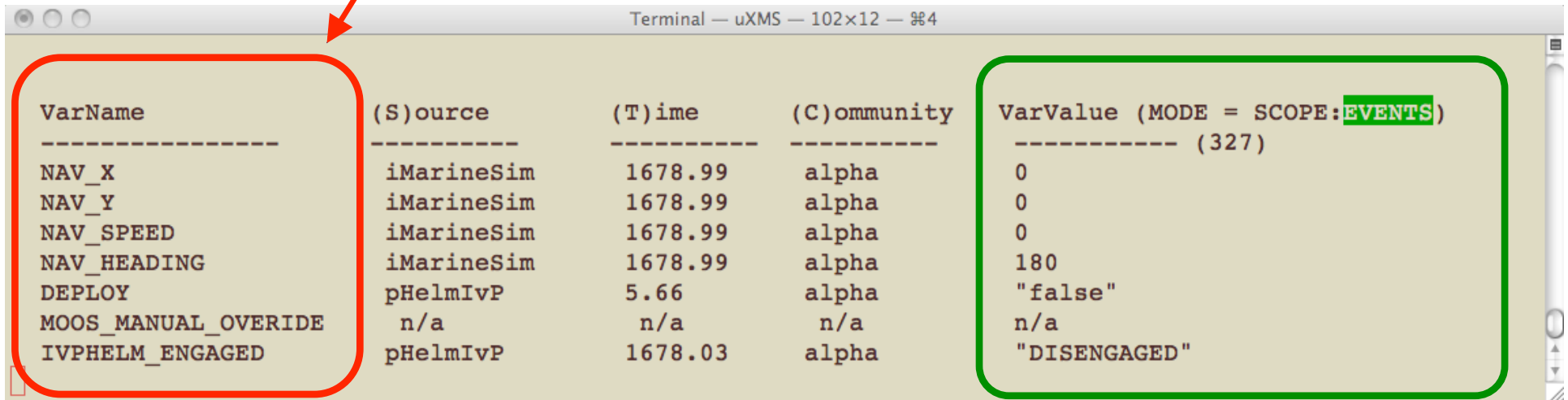
The uXMS Utility:
Specifying the Variables to be Scoped

The variables to be scoped are given on the command line:

```
$ uXMS alpha.moos NAV_X NAV_Y NAV_SPEED NAV_HEADING DEPLOY
                 MOOS_MANUAL_OVERIDE DEPLOY IVPHELM_ENGAGED
```

Each report dedicates a line to each variable:
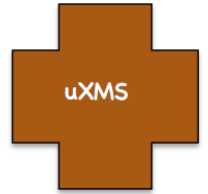
```
Terminal — uXMS — 102×12 — ⌘4

VarName              (S)ource        (T)ime      (C)ommunity    VarValue (MODE = SCOPE:EVENTS)
----------------     ----------      ----------  ----------     ----------- (327)
NAV_X                iMarineSim      1678.99     alpha          0
NAV_Y                iMarineSim      1678.99     alpha          0
NAV_SPEED            iMarineSim      1678.99     alpha          0
NAV_HEADING          iMarineSim      1678.99     alpha          180
DEPLOY               pHelmIvP        5.66        alpha          "false"
MOOS_MANUAL_OVERIDE   n/a             n/a         n/a           n/a
IVPHELM_ENGAGED      pHelmIvP        1678.03     alpha          "DISENGAGED"
```

The variable values are shown in the fifth and last column:
The variable type (string or double) is indicated by quoting the string values
The value of "n/a" indicates the variable has never been written to.

Mike Benjamin

The uXMS Utility:
Components of the scope report

The IP address and port number of the MOOSDB is in the .moos file on the command line:

```
$ uXMS alpha.moos NAV_X NAV_Y NAV_SPEED NAV_HEADING DEPLOY
                  MOOS_MANUAL_OVERIDE DEPLOY IVPHELM_ENGAGED
```
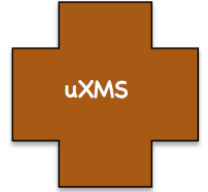
The 2nd column of each report line shows the *source* of the variable posting:
(The MOOS Application that last published the variable)

```
Terminal — uXMS — 102×12 — ⌘4

VarName              (S)ource      (T)ime      (C)ommunity   VarValue (MODE = SCOPE:EVENTS)
----------------    ----------    ----------   ----------    ----------- (327)
NAV_X                iMarineSim    1678.99      alpha         0
NAV_Y                iMarineSim    1678.99      alpha         0
NAV_SPEED            iMarineSim    1678.99      alpha         0
NAV_HEADING          iMarineSim    1678.99      alpha         180
DEPLOY               pHelmIvP      5.66         alpha         "false"
MOOS_MANUAL_OVERIDE   n/a           n/a          n/a          n/a
IVPHELM_ENGAGED      pHelmIvP      1678.03      alpha         "DISENGAGED"
```

The third column shows the time at which the last posting to the variable was made.

The fourth column shows the name of the MOOS community of the source application.

Mike Benjamin

The uXMS Utility:
Scoping on the Alpha Example Mission
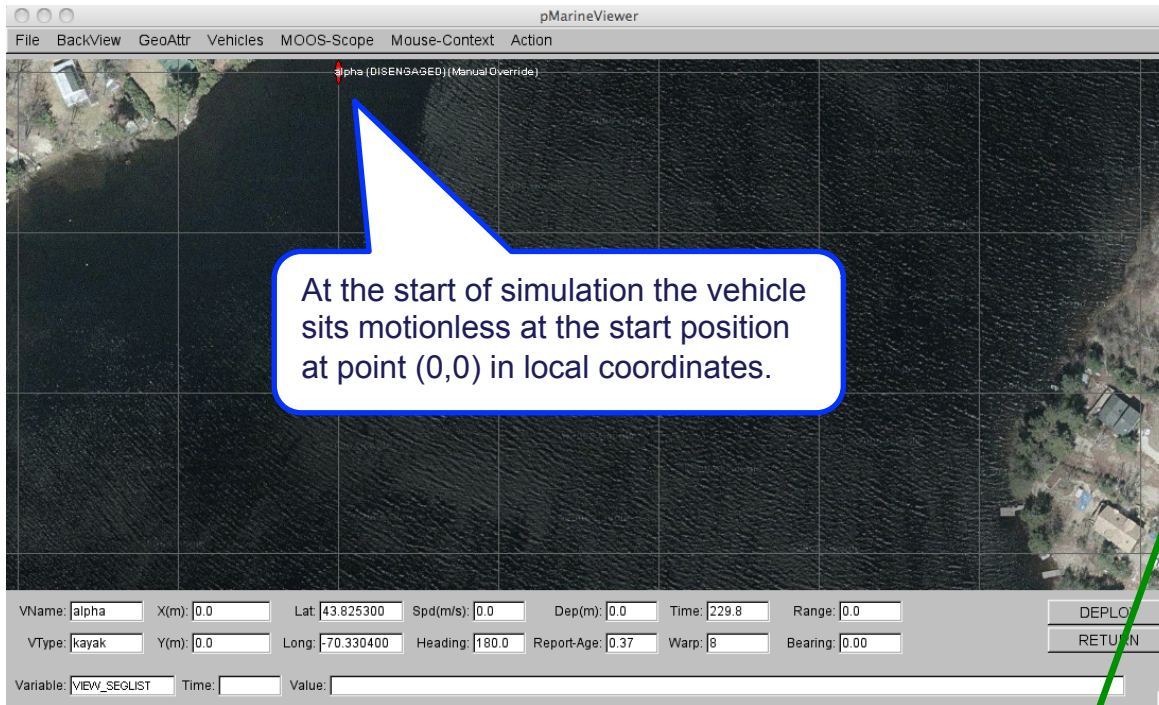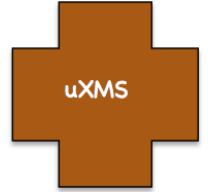
At the outset the vehicle is sitting motionless as shown in the MOOS variables
- `NAV_X`
- `NAV_Y`
- `NAV_SPEED`
- `NAV_HEADING`

At the start of simulation the vehicle sits motionless at the start position at point (0,0) in local coordinates.

The helm is "DISENGAGED" as evidenced by the MOOS variable `IVPHELM_ENGAGED`

Mike Benjamin

The uXMS Utility:
Scoping on the Alpha Example Mission

The vehicle is under way after hitting the DEPLOY button.

The DEPLOY button is configured to post:
DEPLOY=true
MOOS_MANUAL_OVERIDE=false

The postings can be seen in the movement of the vehicle and in the uXMS report:

The uXMS Utility:
Refresh Modes

The uXMS *refresh mode* determines when a new report is written to the console.

- PAUSED mode: A new report will not be written until user requests it.
- EVENTS mode: A new report is written when a scoped variables changes.
- STREAMING mode: A new report is written on each uXMS iteration.

Refresh Modes
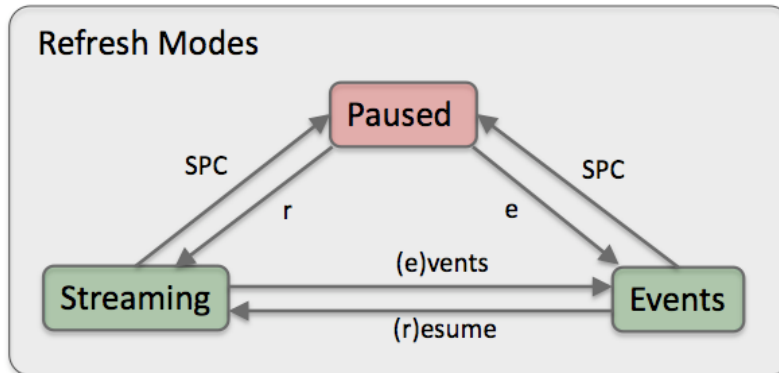
The modes may be switched at the console:

- SPACEBAR – pauses the scope and requests a single new report.
- 'e' or 'E' – moves the scope into Events mode.
- 'r' or 'R' – moves the scope into Streaming mode. ('s' is reserved for something else).

- The PAUSED mode is a key feature of uXMS – it minimizes communications bandwidth.

- The default mode is the EVENTS mode.

- The mode at launch time may be change by specifying '--mode=paused' on command line.

Mike Benjamin

The uXMS Utility:
Refresh Mode Indicator

The uXMS *refresh mode* is indicated in the top right-hand corner of each report:

STREAMING

PAUSED

EVENTS

The uXMS Utility:
Content Modes

The uXMS *content mode* determines what is written in the reports to the console.

- SCOPING mode: A report contains a snapshot of variables in the scope list.
- HISTORY mode: A report contains the recent history of given variable.
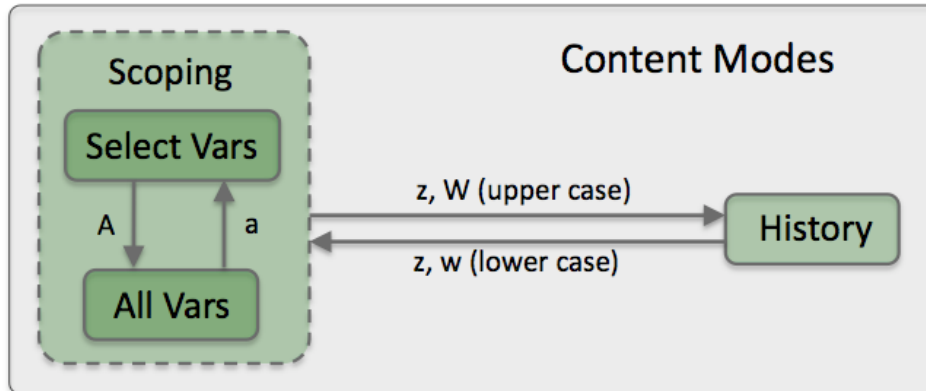
The modes may be switched at the console:

- 'W' always puts uXMS into History mode.
- 'w' always puts uXMS into Scoping mode.
- 'z' or 'Z' – toggles uXMS between Scoping and History mode.

The SCOPING has two sub-modes:

- The "SelectVars" mode reports only on variables in the configured scope list.

- The "AllVars" mode reports on ALL variables known to the MOOSDB.

The modes may be switched at the console:

- 'A' always puts uXMS into SelectVars Scoping mode.
- 'a' always puts uXMS into AllVars Scoping mode.

The uXMS Utility:
The "History" Content Mode

The uXMS *History* content mode shows the recent history of a specified MOOS variable.

- The history is limited to 20 lines (configurable)
- The refresh mode may also be set to EVENTS, PAUSED or STREAMING.

Successive duplicate entries are condensed into a single line with the number of duplicates indicated in parentheses.

Mike Benjamin

The uPokeDB Utility:
Poking the MOOSDB from the Console

MOOS Modules:

- uXMS -   A tool for focused scoping of the MOOSDB from the console.

- uPokeDB -   A tool for poking the MOOSDB from the command line.

- pMarineViewer -   A GUI tool for rendering vehicle operations onto an geo-referenced display.

- pNodeReporter -   Captures vehicle state information and publishes a summary string.

- uHelmScope -   A specialized scope on IvP Helm status and recent history.

- pBasicContactMgr -   A simple manager of vehicle contacts, and generation of alerts.

- uTimerScript -   A tool for scripting (possibly conditional and random) pokes to the MOOSDB.

The uPokeDB Utility:
Definition of a MOOS Poke

What is a MOOS *poke*?

A poke is the publication of a MOOS variable-value pair to a given MOOSDB.
The "poke" implies that publication is "one-time" event.
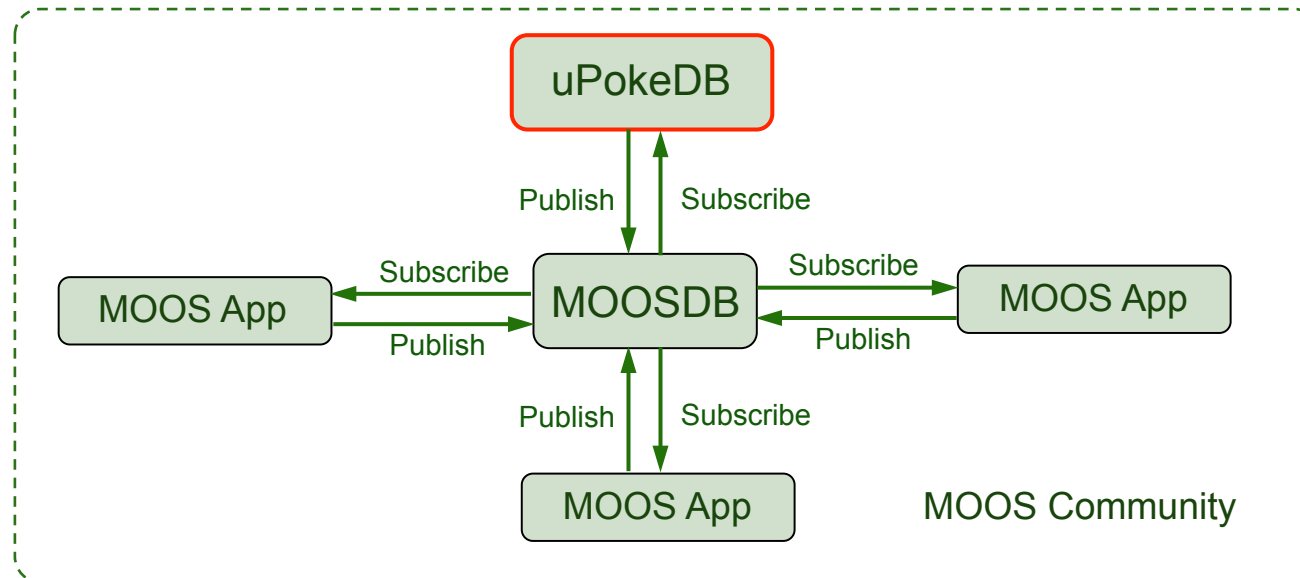
- uPokeDB primarily *publishes* to the MOOSDB (the poke).
- uPokeDB also *subscribes* to the MOOSDB for mail on the variable it is poking – to show the user the variable value prior to the poke, and confirm the variable value after the poke.

The uPokeDB Utility:
Launching the Utility and Understanding the Output

The uPokeDB utility is launched from the command line:

```
$ uPokeDB alpha.moos DEPLOY=true MOOS_MANUAL_OVERRIDE=false
```

```
 ● ● ●                Terminal — tcsh — 76×35 — ⌘6
$ uPokeDB alpha.moos DEPLOY=true MOOS_MANUAL_OVERRIDE=false
Mission File was provided: alpha.moos
***************************************************
*                                                 *
*       This is MOOS Client                       *
*       c. P Newman 2001                          *
*                                                 *
***************************************************

----------------MOOS CONNECT---------------------
  contacting a MOOS server localhost:9000 -  try 00001
  Contact Made
  Handshaking as "uPokeDB"
  Handshaking Complete
  Invoking User OnConnect() callback...ok
--------------------------------------------------

uPokeDB is Running:
        AppTick   @ 5.0 Hz
        CommsTick @ 5 Hz

PRIOR to Poking the MOOSDB
  VarName               (S)ource    (T)ime     VarValue
  ---------------       ----------  ---------  -------------
  DEPLOY                pHelmIvP    0.78        "false"
  MOOS_MANUAL_OVERRIDE

AFTER Poking the MOOSDB
  VarName               (S)ource    (T)ime     VarValue
  ---------------       ----------  ---------  -------------
  DEPLOY                uPokeDB     7.23        "true"
  MOOS_MANUAL_OVERRIDE  uPokeDB     7.23        "false"
Mike B $
```

(1) Upon launching uPokeDB will confirm connection to the MOOSDB.

(2) It will write to stdout, the current values of the variables being poked.

(3) It will confirm the new values of the variables after the poke – as seen by the mail received by the MOOSDB.

(4) It will then quit.
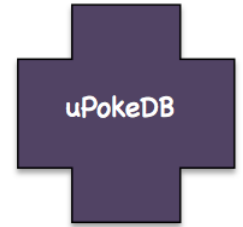
The uPokeDB Utility:
Other ways of Poking the MOOSDB

- A "Poke" is just publication to the MOOSDB, no different than the publications that occur when a MOOS application publishes/writes/posts to the MOOSDB.

- A publication is only "poke" because it is regarded as being outside the "normal" set of variables published by that particular application.

Some other utilities and methods for Poking The MOOSDB:

(1) iRemote:
  - It may be configured to associate a poke with any unmapped key.

(2) uTermCommand:
  - A utility for configuring user-defined pokes (variable-value pairs) with a unique key word. uTermCommand then allows the user to type in the key word and trigger the poke. The key word may trigger more than one poke if desired.
  - The uTermCommand utility is in the moos-ivp tree.

(3) pMarineViewer:
  - On-screen buttons may be configured to trigger one or more user-defined pokes.
  - An "Action" pull-down menu may be configure to associate a pull-down menu item with one or more pokes.

The pMarineViewer Utility:
A GUI for Mission Control

MOOS Modules:

- uXMS -   A tool for focused scoping of the MOOSDB from the console.
- uPokeDB -   A tool for poking the MOOSDB from the command line.
- pMarineViewer -   A GUI tool for rendering vehicle operations onto an geo-referenced display.
- pNodeReporter -   Captures vehicle state information and publishes a summary string.
- uHelmScope -   A specialized scope on IvP Helm status and recent history.
- pBasicContactMgr -   A simple manager of vehicle contacts, and generation of alerts.
- uTimerScript -   A tool for scripting (possibly conditional and random) pokes to the MOOSDB.
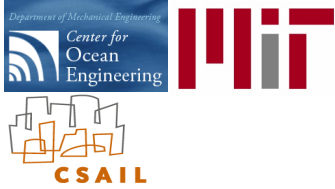
The pMarineViewer Utility:
What it is, and is not
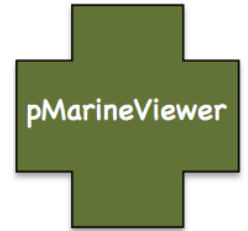
What is pMarineViewer?

- A run-time tool for rendering one or more vehicles during operation or simulation.

- Rendering is possible on a geographical map, given a map image and coordinates.

- Geometric objects, e.g., a set of waypoints or polygon, may also be be rendered.

- It may be used for command-and-control by configuring pokes to the local MOOSDB.

What pMarineViewer is NOT:

- It is not a mission-planning tool.

- It is not a post-mission analysis tool (unless using the uPlayback utility).

- It is not capable of "pausing" or moving back in time.

- It does not have any communications capability to other MOOS communities, local or remote.

The pMarineViewer Utility:
One Simple MOOS Community Topology

A simple topology, used in the Alpha example mission:

- pMarineViewer connects to the same MOOSDB (MOOS Community) running the vehicle.
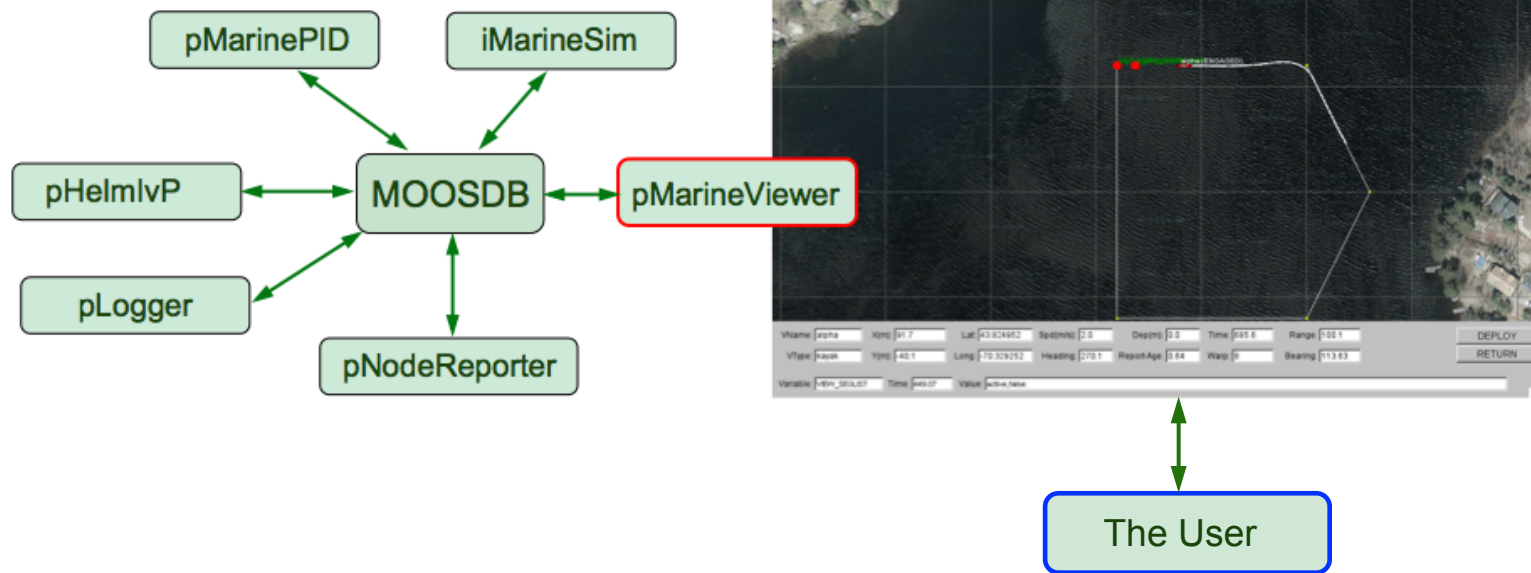
- The User interacts with the GUI to alter the rendering perspective and poke commands to the MOOSDB.

- pNodeReporter publishes NODE_REPORT postings, read by pMarineViewer to update vehicle positions.

- pHelmIvP publishes geometric artifacts like waypoints, read by pMarineViewer and rendered.

The pMarineViewer Utility:
A More Extendable MOOS Community Topology

pMarineViewer

A simple topology, used in the Alpha example mission:

- pMarineViewer runs in its own dedicated MOOS Community – typically on a different machine.

In-Field | Top-side/Shore-side

- The Comms connection may be Acomms, Wifi, Iridium, or just may be running on the same machine.

- There may be an arbitrary number of vehicles connected to the pMarineViewer Community.

The pMarineViewer Utility:
A More Extendable MOOS Community Topology

pMarineViewer

A simple topology, used in the Alpha example mission:

• pMarineViewer runs in its own dedicated MOOS Community – typically on a different machine.

• The Comms connection may be Acomms, Wifi, Iridium, or just may be running on the same machine.

• There may be an arbitrary number of vehicles connected to the pMarineViewer Community.

The pMarineViewer Utility:
The BackView Pull-Down Menu

The pMarineViewer Utility:
The GeoAttributes Pull-Down Menu

The pMarineViewer Utility:
The GeoAttributes Pull-Down Menu

Toggle or adjust default attributes for the XYPolygon, XYSegList, XYPoint, and XYGrid objects.

Toggle the rendering of the Datum

Toggle or adjust default attributes for Markers, OpArea and DropPoint objects.

XYSegList

XYPoints

The pMarineViewer Utility:
The Vehicles Pull-Down Menu

Toggle off/on the rendering of vehicles

Adjust which vehicle is "active". This vehicle has its info displayed in the fields at the bottom of the viewer

Pan the viewer to the vehicle

Toggle off/on the rendering vehicle trails and adjust length

Change default color of vehicles

The pMarineViewer Utility:
The MOOS-Scope Pull-Down Menu

The pNodeReporter Utility:
Summarizing a Node's Status

MOOS Modules:

- uXMS -   A tool for focused scoping of the MOOSDB from the console.

- uPokeDB -   A tool for poking the MOOSDB from the command line.

- pMarineViewer -   A GUI tool for rendering vehicle operations onto an geo-referenced display.

- pNodeReporter -   Captures vehicle state information and publishes a summary string.

- uHelmScope -   A specialized scope on IvP Helm status and recent history.

- pBasicContactMgr -   A simple manager of vehicle contacts, and generation of alerts.

- uTimerScript -   A tool for scripting (possibly conditional and random) pokes to the MOOSDB.

The pNodeReporter Utility:
What it is, and is not

What is pNodeReporter?

- A run-time tool for gathering information about ownship and summarizing in a single MOOS variable.

- The single MOOS variable is NODE_REPORT_LOCAL.

- It reports information on the platform position and trajectory.

- It reports information on the platform type, and length.

- It reports certain key information regarding the state of the IvP Helm.

- It can be viewed as a loose proxy for an AIS (Automatic Information System) report.

What pNodeReporter is NOT:

- It does not handle communications between platforms.

- It does not handle incoming reports from other platforms.

The pNodeReporter Utility:
Basic Functions

Source	Field	Example

Derived from the MOOS Community name.

NAME — "NAME=alpha"

Derived from pNodeReporter configuration parameters

TYPE — "TYPE=UUV, LENGTH=4.6"
LENGTH

Derived from DB_UPTIME

MOOSDB_TIME — "MOOSDB_TIME=146.7"

Derived from calling MOOSTime() from within pNodeReporter

UTC_TIME — "UTC_TIME=1281548666.57"

X
Y
HDG
SPD
YAW
DEPTH
LAT
LON

Derived from the vehicle navigation system:
NAV_X, NAV_Y etc.

"X=109.2,Y=-22.0,HDG=37.2,
SPD=1.7,YAW=0.78,DEPTH=12,
LAT=43.82530,LON=-70.33040"

Derived from output of the helm:
IVPHELM_ENGAGED,
IVPHELM_SUMMARY

MODE — "MODE=MODE@ACTIVE:SURVEY,
ALLSTOP=clear

Mike Benjamin

The pNodeReporter Utility:
Alpha Example Mission

In the Alpha Example Mission:
- Launch the mission
- Run uXMS with:

```
$ uXMS —history=NODE_REPORT_LOCAL
```

- Note the successive values of NODE_REPORT_LOCAL reported.

The pNodeReporter Utility:
The Optional Blackout Interval Option

Normally a node report once per iteration, determined solely by the APP_TICK parameter.
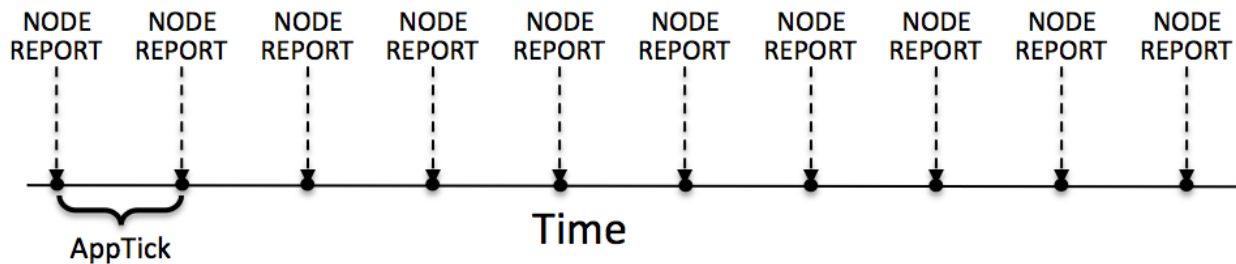
At times it is useful to add an artificial delay between postings.   `BLACKOUT_INTERVAL = 35`
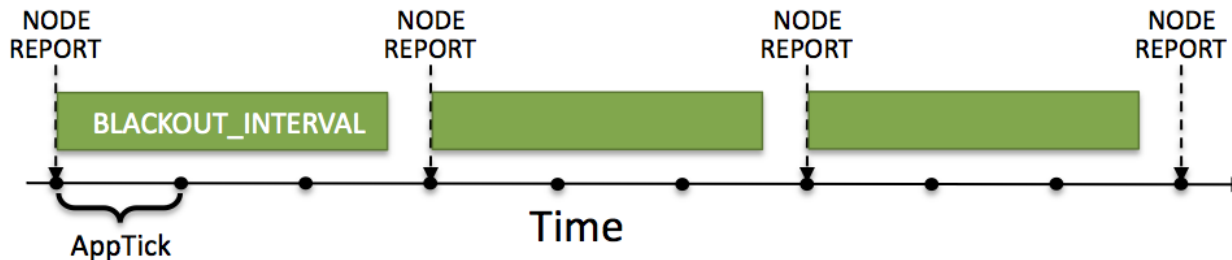
The pNodeReporter Utility:
Random Blackout Intervals

- Node reports are typically only useful as information sent to other nodes.
- There are often dropped node messages due to the uncertain nature of communications.
- Applications receiving node reports usually implement provisions that take dropped messages into account.
- For example, a collision avoidance behavior may extrapolate the contact position in between node reports.
- To test the robustness of dealing with dropped node reports, we want to simulate them easily.

- The dropouts occur in the field more or less randomly (but may be range dependent etc.)

The Blackout Interval may be configured to vary randomly: `BLACKOUT_VARIANCE = 45`
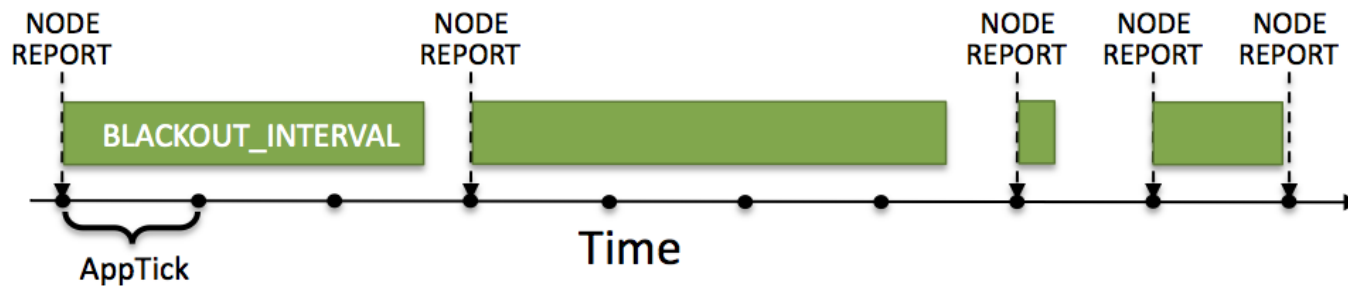
The uHelmScope Utility:
Scoping on the IvP Helm

uHelmScope

MOOS Modules:

- uXMS -   A tool for focused scoping of the MOOSDB from the console.
- uPokeDB -   A tool for poking the MOOSDB from the command line.
- pMarineViewer -   A GUI tool for rendering vehicle operations onto an geo-referenced display.
- pNodeReporter -   Captures vehicle state information and publishes a summary string.
- uHelmScope -   A specialized scope on IvP Helm status and recent history.
- pBasicContactMgr -   A simple manager of vehicle contacts, and generation of alerts.
- uTimerScript -   A   tool for scripting (possibly conditional and random) pokes to the MOOSDB.

The uHelmScope Utility:
What it is, and is not

What is uHelmScope?

- It is a specialized scope on the MOOSDB for reporting information specific to the IvP Helm.

- It is console-based (like uXMS) and requires no graphics libraries.

- It reports on which behaviors are active, running, idle and complete.

- It reports the helm decision for each decision variable.

- It is capable of pausing and stepping back and forth in time.

- It includes a generic MOOS scope for convenience.

- It reports on which variables are posted by the helm on a given iteration.

What uHelmScope is NOT:

- It does not post any information to the helm or any other MOOS applications.

- It is not a graphical tool.

The uHelmScope Utility:
A typical report from uHelmScope

From the Alpha example mission, shortly after deploying the vehicle:

```
Terminal — uHelmScope — 127×38 — ⌘6

=============    uHelmScope Report   =============  ENGAGED   (2)
  Helm Iteration: 403      (hz=0.25)(5)   (hz=0.25)(18)   (hz=0.26)(max)
  IvP functions:  1
  Mode(s):
  SolveTime:       0.00     (max=0.00)
  CreateTime:      0.00     (max=0.01)
  LoopTime:        0.00     (max=0.01)
  Halted:          false    (0 warnings)
Helm Decision: [speed,0,4,21] [course,0,359,360]
  course = 73.0
  speed = 2.0
Behaviors Active: ---------- (1)
  waypt_survey (100.9) (pwt=100.00) (pcs=6) (cpu=0.43) (upd=0/0)
Behaviors Running: --------- (0)
Behaviors Idle: ----------- (2)
  waypt_return, hsline
Behaviors Completed: ------- (0)


#  MOOSDB-SCOPE ----------------------------------- (Hit '#' to en/disable)
#
#  VarName          Source       Time      Community  VarValue
#  ---------------  -----------  -------   ---------  -----------
#  BHV_WARNING        n/a          n/a       n/a        n/a
#  DEPLOY*          pMarin..ewer 128.28    alpha      "true"
#  HSLINE*          pHelmIvP     128.28    alpha      "off"
#  RETURN*          pMarin..ewer 128.28    alpha      "false"


@  BEHAVIOR-POSTS TO MOOSDB ---------------------- (Hit '@' to en/disable)
@
@  MOOS Variable     Value
@  -------------     -------   (BEHAVIOR=waypt_survey)
@  WPT_STAT          vname=alpha,behavior-name=waypt_survey,index=2,hit +
@  VIEW_POINT        active,true:label,alpha_waypoint:label_color,0.000 +
@  VIEW_POINT        active,true:label,alpha_track-point:label_color,0. +
```

The *Helm Report*: An overview of which behaviors' run state and information on IvP function characteristics and solve time.

The *MOOS Scope*: A mini MOOS scope for the convenience of scoping on a few variables of the user's choosing.

The Behavior-Posts: A set of variable-value pairs posted by the helm in the current iteration.

The uHelmScope Utility:
A Closer Look at the Helm Report (the top section)

uHelmScope

# of IvP functions in current decision

Engagement status

# of reports written to the console

```
==============    uHelmScope Report    =============== ENGAGED  (2)
  Helm Iteration: 403      (hz=0.25)(5)   (hz=0.25)(18)   (hz=0.26)(max)
  IvP functions:  1
  Mode(s):
  SolveTime:       0.00     (max=0.00)
  CreateTime:      0.00     (max=0.01)
  LoopTime:        0.00     (max=0.01)
  Halted:          false    (0 warnings)
Helm Decision: [speed,0,4,21] [course,0,359,360]
  course = 73.0
  speed = 2.0
Behaviors Active: ---------- (1)
  waypt_survey (100.9) (pwt=100.00) (pcs=6) (cpu=0.43) (upd=0/0)
Behaviors Running: --------- (0)
Behaviors Idle: ----------- (2)
  waypt_return, hsline
Behaviors Completed: ------- (0)
```

Average CPU time between iterations for the last 5 iterations

Maximum CPU time observed for all iterations.

Mike Benjamin

The uHelmScope Utility:
A Closer Look at the Helm Report (the top section)

uHelmScope

# of IvP functions in current decision

Engagement status

# of reports written to the console

```
==============     uHelmScope Report   ============== ENGAGED  (2)
  Helm Iteration: 403      (hz=0.25)(5)  (hz=0.25)(18)  (hz=0.26)(max)
  IvP functions:  1
  Mode(s):
  SolveTime:       0.00    (max=0.00)
  CreateTime:      0.00    (max=0.01)
  LoopTime:        0.00    (max=0.01)
  Halted:          false   (0 warnings)
Helm Decision: [speed,0,4,21] [course,0,359,360]
● course = 73.0
● speed = 2.0
Behaviors Active: ---------- (1)
   waypt_survey (100.9) (pwt=100.00)
Behaviors Running: --------- (0)
Behaviors Idle: ------------ (2)
   waypt_return, hsline
Behaviors Completed: ------- (0)
```

Total *solve* time for the current iteration – and max solve time for all iterations.

Total *create* time for the current iteration – and max solve time for all iterations.

The Helm decision space: variable name, low value, high value and number of points.

The current helm decision.

Mike Benjamin

MOOS-DAWG 2011

The uHelmScope Utility:
A Closer Look at the Helm Report (the top section)

```
==============    uHelmScope Report   ==============  ENGAGED   (2)
  Helm Iteration: 403       (hz=0.25)(5)   (hz=0.25)(18)   (hz=0.26)(max)
  IvP functions:  1
  Mode(s):
  SolveTime:        0.00     (max=0.00)
  CreateTime:       0.00     (max=0.01)
  LoopTime:         0.00     (max=0.01)
  Halted:           false    (0 warnings)
Helm Decision: [speed,0,4,21] [course,0,359,360]
  course = 73.0
  speed = 2.0
Behaviors Active: ---------- (1)
  waypt_survey (100.9) (pwt=100.00) (pcs=6) (cpu=0.43) (upd=0/0)
Behaviors Running: --------- (0)
Behaviors Idle: ------------ (2)
  waypt_return, hsline
Behaviors Completed: ------- (0)
```

Total behaviors in this run state

CPU time to make the IvP Function

Priority Weight

Pieces in the IvP Function

# of successful updates vs. # of attempted behavior updates.

Behavior States: Which behaviors are active, running, idle or completed. For active behaviors, information is given on their IvP function.

The uHelmScope Utility:
A Closer Look at the MOOS Scope (middle section)

```
#  MOOSDB-SCOPE ----------------------------------------  (Hit '#' to en/disable)
#
#  VarName            Source        Time      Community  VarValue
#  ---------------    ----------    -------   ---------  -----------
#  BHV_WARNING         n/a           n/a        n/a      n/a
#  DEPLOY*            pMarin..ewer  128.28      alpha     "true"
#  HSLINE*            pHelmIvP      128.28      alpha     "off"
#  RETURN*            pMarin..ewer  128.28      alpha     "false"
```

List of variables to scope

Source of the last post

Time of the last post

Variable Value

Mike Benjamin

The uHelmScope Utility:
A Closer Look at the MOOS Scope (middle section)

uHelmScope

The *Behavior-Posts* section displays only those variable-value pairs posted by the Helm on the current iteration.

```
@   BEHAVIOR-POSTS TO MOOSDB ---------------------- (Hit '@' to en/disable)
@
@   MOOS Variable         Value
@   -------------         -------   (BEHAVIOR=waypt_survey)
@   WPT_STAT              vname=alpha,behavior-name=waypt_survey,index=2,hit +
@   VIEW_POINT            active,true:label,alpha_waypoint:label_color,0.000 +
@   VIEW_POINT            active,true:label,alpha_track-point:label_color,0. +
```

Variables posted   Variable Values

The uHelmScope Utility:
Examining the Helm Hierarchical Mode Declarations

The Hierarchical Mode Declarations for Henry in the Berta Example Mission:

```
// Excerpt from the
// henry.bhv file.

Set MODE = Active {
  DEPLOY = true
} Inactive

Set MODE = STATION-KEEPING {
  MODE = ACTIVE
  STATION_KEEP = true
}

Set MODE = RETURNING {
  MODE = ACTIVE
  RETURN = true
}

Set MODE = LOITERING {
  MODE = ACTIVE
  LOITER = true
}
```

Text File Configuration
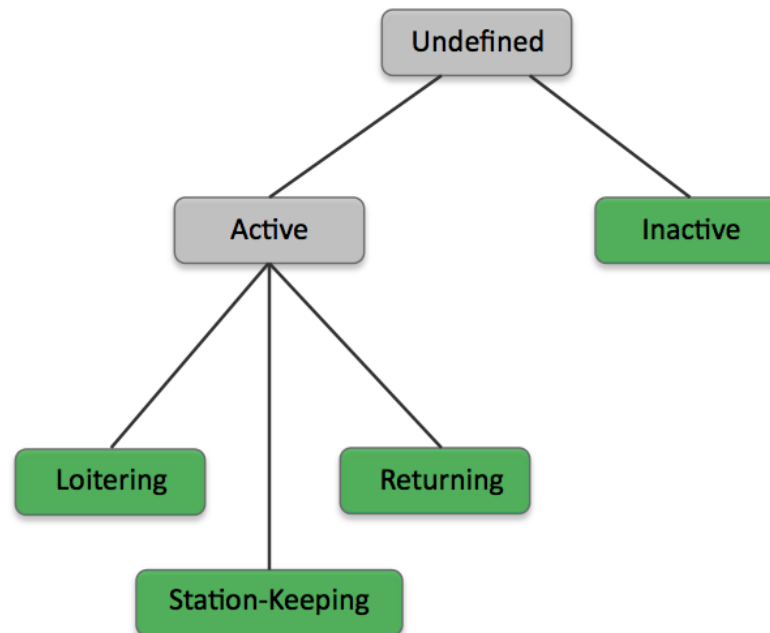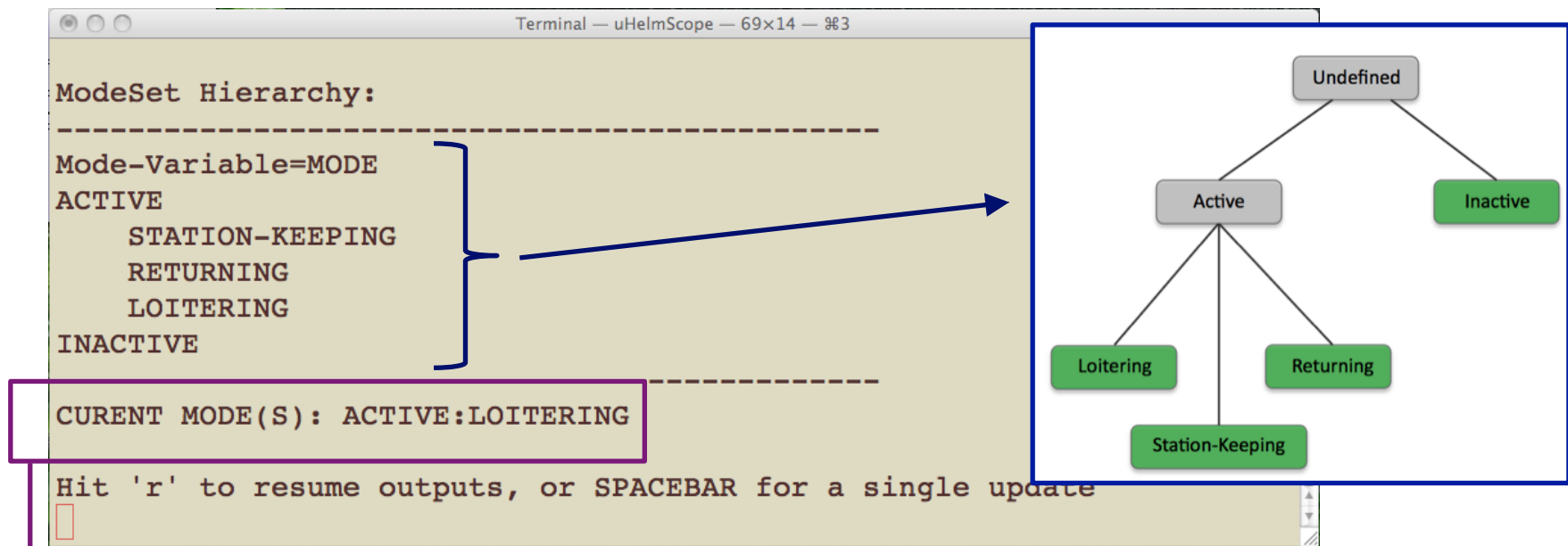
Graphical representation (manually generated)

uHelmScope can used to visually confirm the configuration matches the intention.

Mike Benjamin

The uHelmScope Utility:
Examining the Helm Hierarchical Mode Declarations

uHelmScope

The hierarchical mode declarations may be viewed by toggling with the 'M' key:


```
Terminal — uHelmScope — 69×14 — ⌘3

ModeSet Hierarchy:
---------------------------------------------------------
Mode-Variable=MODE
ACTIVE
      STATION-KEEPING
      RETURNING
      LOITERING
INACTIVE
                                        ---------------
CURENT MODE(S): ACTIVE:LOITERING

Hit 'r' to resume outputs, or SPACEBAR for a single update
```

The prevailing helm mode is also shown in this screen.

Mike Benjamin

The pBasicContactMgr Utility:
Managing Platform Contacts

MOOS Modules:

- uXMS - A tool for focused scoping of the MOOSDB from the console.

- uPokeDB - A tool for poking the MOOSDB from the command line.

- pMarineViewer - A GUI tool for rendering vehicle operations onto an geo-referenced display.

- pNodeReporter - Captures vehicle state information and publishes a summary string.

- uHelmScope - A specialized scope on IvP Helm status and recent history.

- pBasicContactMgr - A simple manager of vehicle contacts, and generation of alerts.

- uTimerScript - A tool for scripting (possibly conditional and random) pokes to the MOOSDB.

The pBasicContactMgr Utility:
What it is, and is not

What is pBasicContactMgr?

- A tool for managing node reports and generating conditional events.

- It posts summary reports for all known contacts.

- It posts events, i.e., alerts, about contacts based on the range to the contact.

- Designed with the IvP Helm in mind to allow the helm to spawn contact-related behaviors dynamically as they become known.

What pBasicContactMgr is NOT:

- It is not a sensor application.

- It does not perform sensor fusion.

- It does not represent or reason about areas of uncertainty associated with contact position.
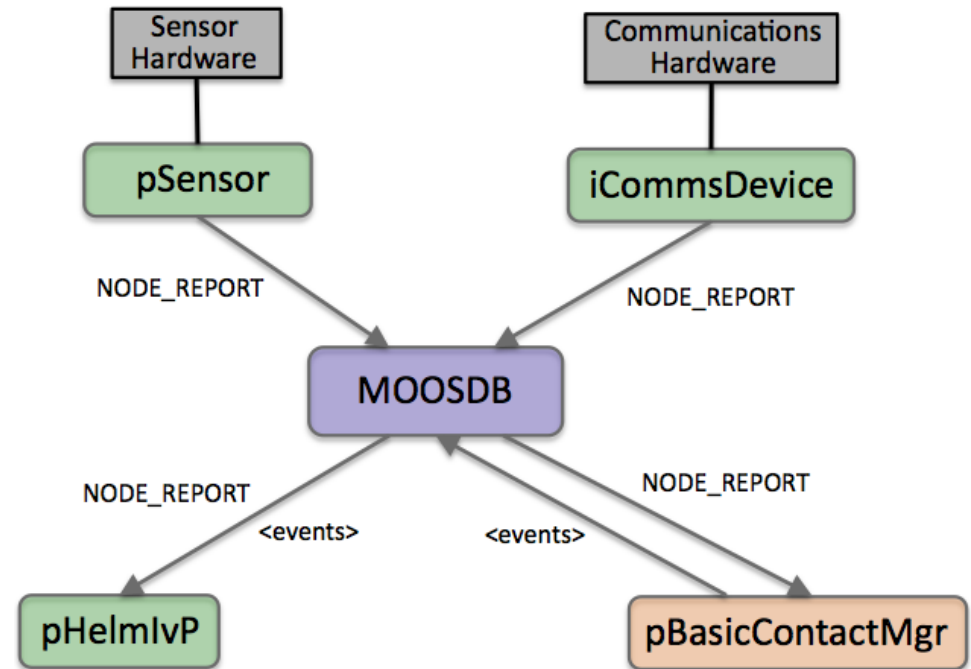
Variables Published:

- CONTACTS_LIST
- CONTACTS_RECAP
- CONTACT_ALERTED
- CONTACTS_UNALERTED
- CONTACTS_RETIRED
- CONTACT_MGR_WARNING

Mike Benjamin

The pBasicContactMgr Utility:
Alerts

What is an Alert?

- It is a posting to the MOOSDB – A MOOS variable-value pair.
- Alerts are generated for a given contact, when the contact is within a given range.
- The value of the alert is configured by the user in the pBasicContactMgr configuration block.

How are they used?

- Alerts may be used to trigger other processes.
- Alerts may also be used for marking an event to be logged and later referenced.

An example (collision avoidance):

- An alert is generated when contact gets "too close".
- The helm is configured with a collision avoidance behavior "template".
- The template is instantiated with a new behavior instance when it receives the alert.

The pBasicContactMgr Utility:
Alert Configuration

Alerts are configured in the MOOS configuration file:

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

The <alert-content> may be any string, including certain macros for expansion. For example:

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

Macros available are:

(many of the same fields found in the node reports, `NODE_REPORT`)

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

The pBasicContactMgr Utility:
Alert Configuration

Alerts are configured in the MOOS configuration file:

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

The <alert-content> may be any string, including certain macros for expansion. For example:

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

Accommodates a helm behavior configuration for dynamic behavior spawning.
See the Berta example mission.

Macros available are:

(many of the same fields found in the node reports, `NODE_REPORT`)

$[VNAME]: The name o...
$[X]: The position...
$[Y]: The position...
$[LAT]: The latitude...
$[LON]: The longitud...
$[HDG]: The reported...
$[SPD]: The reported...
$[DEP]: The reported depth of the contact.
$[VTYPE]: The reported vessel type of the contact.
$[UTIME]: The UTC time of the last report for the contact.

The pBasicContactMgr Utility:
Alert Triggers

Alerts are triggered by range. Configured in the MOOS configuration file:

```
    ALERT_RANGE = <distance>    // meters
ALERT_CPA_RANGE = <distance>    // meters
 ALERT_CPA_TIME = <duration>    // seconds
```

`ALERT_RANGE` – when a contact is within this range an alert is generated.
`ALERT_CPA_RANGE` – when a contact is within this range and its closest point of approach (CPA) is within the alert range, an alert is generated.
`ALERT_RANGE` – The time used for CPA calculation.

Examples:
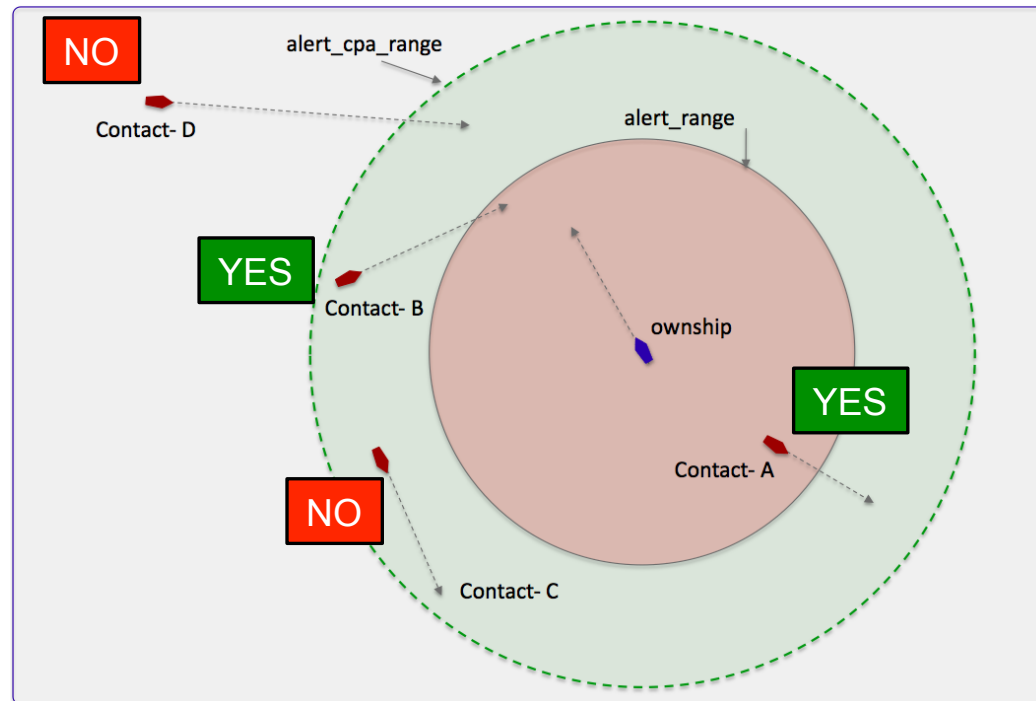
The pBasicContactMgr Utility:
Contacts, Alerts, Record keeping

The following are reported (Posted to the MOOSDB) on each iteration:

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

Examples:

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

The pBasicContactMgr Utility:
Contact Resolution

- An alert is generated by the contact manager for a given contact ONCE (when the trigger criteria is first met).

- Sometimes a consumer of alerts may want to receive additional future alerts should the contact come back into range.

- If pBasicContactMgr receives the message `CONTACT_RESOLVED`, for a given contact, it will generate another alert for that contact should the contact again meet the trigger criteria.

> The contact resolution mechanism is used to handle the scenario where a contact comes into range, exits the range, and later returns.

The pBasicContactMgr Utility:
The Berta Example Mission

The pBasicContactMgr Utility:
The Berta Example Mission

The pBasicContactMgr Utility:
The Berta Example Mission

The uTimerScript Utility:
Scripting Events to the MOOSDB

uTimerScript

MOOS Modules:

- uXMS -   A tool for focused scoping of the MOOSDB from the console.

- uPokeDB -   A tool for poking the MOOSDB from the command line.

- pMarineViewer -   A GUI tool for rendering vehicle operations onto an geo-referenced display.

- pNodeReporter -   Captures vehicle state information and publishes a summary string.

- uHelmScope -   A specialized scope on IvP Helm status and recent history.

- pBasicContactMgr -   A simple manager of vehicle contacts, and generation of alerts.

- uTimerScript -   A tool for scripting (possibly conditional and random) pokes to the MOOSDB.

The uTimerScript Utility:
Overview

uTimerScript

What is uTimerScript?

- A tool that allows the user to script a set of pre-configured events (pokes) to a MOOSDB.

- Each event can be configured to happen after a specified amount of elapsed time.

- Enables us to fake incoming command-and-control messages, sensor events etc.

A simple example:

```
ProcessConfig = uTimerScript
{
  AppTick   = 4
  CommsTick = 4

  EVENT = MOOS_MANUAL_OVERRIDE, false, 10

  EVENT = var=DEPLOY, val=true, time=15
}
```

This simple script will launch the Alpha or Berta missions automatically.

```
EVENT = var<variable>, val=<value>, time=<delay>
```

When does the script start?

- By default the script starts when uTimerScript connects to the MOOSDB and begins to Iterate().

- It may be configured in the "paused" mode

- It may be configured to include a delay once it has started.

- It may be configured to require conditions be met before starting.

Starting the script in the PAUSED mode, with a DELAY.

```
ProcessConfig = uTimerScript
{
  AppTick   = 4
  CommsTick = 4

        EVENT = var=MOOS_MANUAL_OVERRIDE, val=false, time=10

        EVENT = var=DEPLOY, val=true, time=15

  CONDITION = ALPHA != 20

  DELAY_START = 30

  PAUSED = true

}
```

Script will be paused if `ALPHA=20`.
(uTimerScript will register for `ALPHA`).

The script may then be un-paused by posting to the MOOSDB:
`UTS_PAUSE=false`.

The uTimerScript Utility:
Randomizing the Event Times

Random event scheduling:

- Events may be configured to occur at a random time in a given interval.

- Random events are useful in testing the robustness of algorithms in varying situations.

The same example script with events randomized:

```
ProcessConfig = uTimerScript
{
  AppTick   = 4
  CommsTick = 4

  EVENT = var=MOOS_MANUAL_OVERRIDE, val=false, time=10:20

  EVENT = var=DEPLOY, val=true, time=10:20

  PAUSED = true
}
```

Event occurs between
10 and 20 seconds after
the script begins

Event times are chosen with uniform probability.

The uTimerScript Utility:
Repeating the script

uTimerScript

Repeating the script:

- The script may be repeated a fixed number or indefinite number of times.

The same example script with events randomized:

```
ProcessConfig = uTimerScript
{
  AppTick    = 4
  CommsTick = 4

  EVENT = var=MOOS_MANUAL_OVERRIDE, val=false, time=10:20

  EVENT = var=DEPLOY, val=true, time=10:20

    RESET_MAX = 10
  RESET_TIME = 120

    RESET_VAR = UTS_RESET
  DELAY_RESET = 12

    SHUFFLE = true
}
```

Script will run 11 times.

Script will reset after 120 seconds regardless if it is finished.

MOOS variable for receiving reset cues.

# of seconds the script will delay on each reset.

If shuffle is `false`, random timestamps will not be recalculated on each reset.

uTimerScript

Macros:

• Macros are used to fill in variable values with information determined at event posting time.

A Script with a simple macro posting:

```
ProcessConfig = uTimerScript
{
  AppTick   = 4
  CommsTick = 4

  EVENT = var=MOOS_MANUAL_OVERRIDE, val=false, time=10:20

  EVENT = var=DEPLOY, val=true, time=10:20

  EVENT = var=SCRIPT_STARTED, val=$[DBTIME], time=0

}
```

The start time of the script will be posted with the value of DBTIME, the total amount of time the MOOSDB has been up.
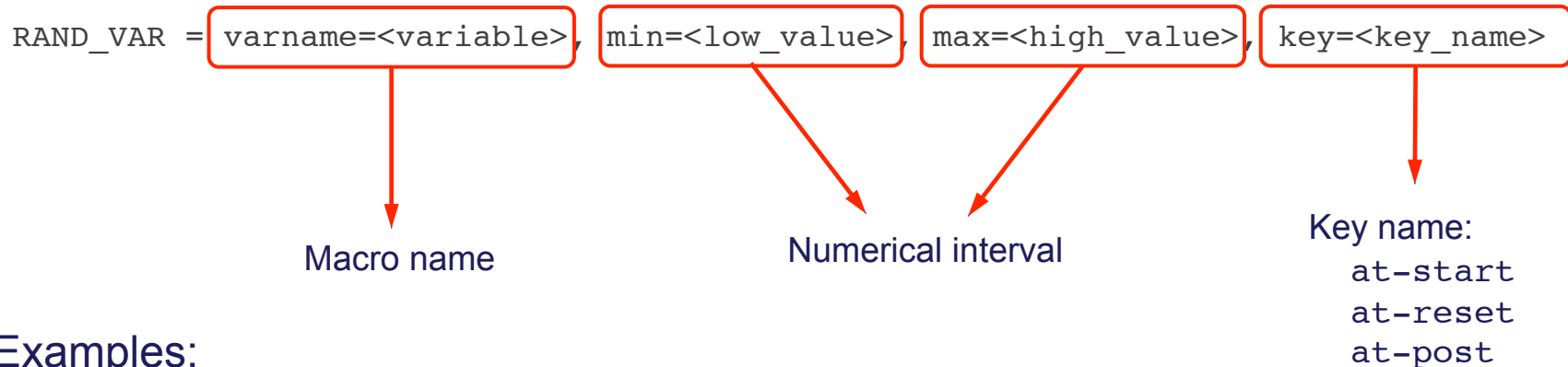
The uTimerScript Utility:
Macro Expansion

uTimerScript
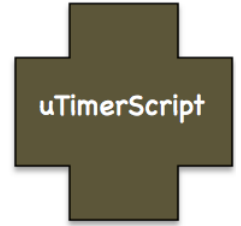
Macros available:

- $[DBTIME]: The estimated amount of time since the MOOSDB started.
- $[UTCTIME]: The UTC time at the time of event posting.
- $[COUNT]: The integer total of all posts thus far in the script – reset to zero on script reset.
- $[TCOUNT]: Same as above except the total is not reset when the script is reset.
- $[IDX]: Similar to $[COUNT], but it expands as a string, "000", "001", "002", etc.

User configured macros with random variables:

```
RAND_VAR = varname=<variable>, min=<low_value>, max=<high_value>, key=<key_name>
```

Macro name

Numerical interval

Key name:
at-start
at-reset
at-post

Examples:

```
RAND_VAR = varname=ANGLE, min=0, max=359, key=at_reset

RAND_VAR = varname=MAGNITUDE, min=0.5, max=1.5, key=at_reset
```

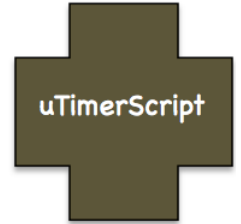# The uTimerScript Utility:
Usage in the Berta Example Mission

Region #1

Region #2

On each iteration of the script, the location of the loiter polygons are randomly set within the two regions.

The uTimerScript Utility:
Usage in the Berta Example Mission

The uTimerScript Utility:
Script Usage in the Berta Example Mission

Permutation of Region locations and loiter assignments

```
ProcessConfig = uTimerScript
{
  AppTick   = 4
  CommsTick = 4

  PAUSED     = false
  RESET_MAX  = unlimited
  RESET_TIME = end

  RANDVAR = varname=X1, min=-25,  max=25,   key=at_reset
  RANDVAR = varname=Y1, min=-100, max=-50,  key=at_reset

  RANDVAR = varname=X2, min=100,  max=150,  key=at_reset
  RANDVAR = varname=Y2, min=-75,  max=-25,  key=at_reset

  EVENT = var=UP_LOITER_2, val="center_assign=$[X1],$[Y1]", time=180
  EVENT = var=UP_LOITER_1, val="center_assign=$[X2],$[Y2]", time=180

  EVENT = var=UP_LOITER_1, val="center_assign=$[X1],$[Y1]", time=360
  EVENT = var=UP_LOITER_2, val="center_assign=$[X2],$[Y2]", time=360
}
```
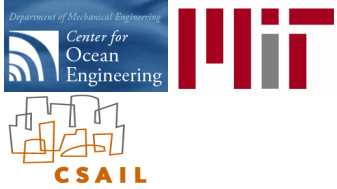
Random variable Macro for Region #1

Region #2

Macro usage in scripted events

The End

Where to find more:

On the web:
   www.moos-ivp.org

Email:
   issues@moos-ivp.org