MOOS-IvP Autonomous Command and Control for Undersea Network Nodes

User's Guide

Henrik Schmidt, Arjuna Balasuriya, and Toby Schneider

[1]Department Mechanical Engineering
Massachusetts Institute of Technology, Cambridge MA

December 8, 2008

Abstract

This paper provides a brief User's Guide for the MIT MOOS-IvP module suite for autonomous communication, command and control AC^3 for aunderwater vehicles operating in an undersea network with limited communication capacity. The software suite implements the command and control of the vehicle using the ASTM F41 proposed standard for autonomous vehicles. Thus, the software architecture incorporates MOOS modules for interfacing the `HelmivP` "backseat driver" autonomy with the lower level vehicle control, handled by the native vehicle autonomy software. The AC^3 infrastructure also includes a software stack for handling the communication with other vehicles and the topside field control using acoustic modems. The combination of the two interface module classes enables the effective vehicle command and control paradigm, where the vehicle following it's initial deployment does not need to surface for new mission assignments, but allows it to remain submerged under autonomous control with mission directives from the topside being passed through the underwater acoustic modem network exclusively. thus, it forms the cornerstone of the Nested Autonomy paradigm for operating distributed undersea networks in severely limited communication environments, as implemented in the ONR UPS concept demonstration, for example, and curently adapted to other undersea surveillance, observation and monitoring programs.

Contents

1 Overview

1.1 Purpose and Scope of this Document

The purpose of this document is to provide a catalog style overview of modules used for the Autonomous Conmmunication, Command and Control of an underwater vehicle, which is part of a distributed undersea network with nested, adaptive and collaborative autonomy. The scope of discussion includes, for each module, a brief description of the module function, authorship, source for download, rough measure of complexity, and module dependencies.

Further, for use by developers of onboard processes and behaviors modules, the description includes a detailed listing and description of MOOS variables published by or subscribed to by each simulator module.

2 MOOS-IvP Communication, Command and Control

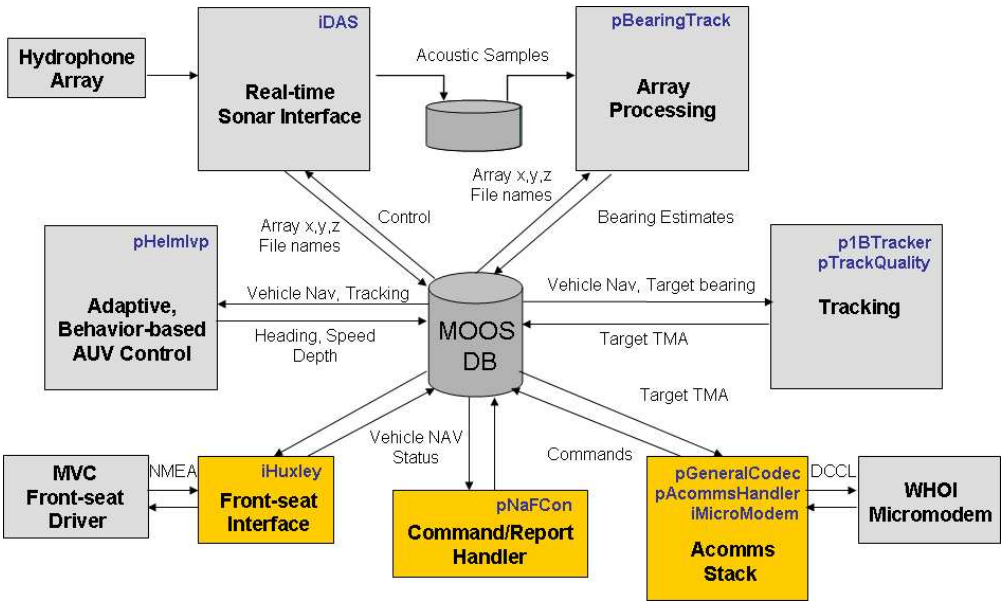

Figure 1: MOOS-IvP community for MIT sonar AUVs, with the autonomous communication, command and control modules highlighted in gold.

The Autonomous Communication, Command and Control components of the MOOS-IvP autonomy system handles the communication between the payload autonomy system, including the autonomous Helm, and the outside world.

The communication with the main vehicle computer "front-seat driver" is handled by the `pHuxley` process, responsible for passing desired speed, heading and depth as desired by the Helm to the main vehicle computer, which is then responsible for translating these commands into actuator actions. The MVC responds back with navigation information, which is then published in the `MOOSDB` for use by other modules, including the autonomy helm `pHelmIvP`.

The connection with the topside field control operators is performed using the acoustic modems,

with the so-called *Communication Stack* handling the coding/decoding (CoDec)message scheduling, and modem transmission and reception.

Both of these interfaces are controlled by the AC^3 handler process, `pNaFCon`, which is responsible for translating command messages from the topside operators into state variables for the inboard autonomy system, and for translating the current vehicle states into status reports for broadcasting to the network. Similarly, the state of the vehicle is translated into commands for the "front-seat driver", such as disabling and enabling the "back-seat driver" control.

A number of other processes are integrated into the prototype autonomy system for managing the state transitions in accordance with the nested autonomy CONOPS.

A list of autonomous communication, command and control processing modules is given in Table 1. A more detailed description is given in the following.

2.1 Node Level Command and Control

In the *Nested Autonomy* paradigm the node level Command and Control is highly tilted towards autonomy, with the vehicle actions being defined by the current vehicle state and the associated behavior set, and the sensor input. The possibility of the cluster and the field control to alter the node behavior is limited to simple, higher level commands, the only role of which is to transition the vehicle to another, allowed state. Thus, for example, if the *NaFCon* operators want to alter the vehicle behavior during a prosecute mission, their only option is to transition the vehicle to another allowed state, defined in the configuration file. The detailed actions such as heading, speed and depth of the node are not directly controllable by the operators. For example, the operators have the power to override an active prosecute mission by transmitting a *Deploy Command* or switching the node to an alternative *Prosecute State*.

Similarly, the state of the node may be altered by other nodes in the cluster only if allowed in the mission configuration on the node. For example, in the MIT Nested Autonomy Prototype, the process `pTargetOpportunity` will react to incoming *Track Reports* by issuing itself a *Prosecute Command* if the target is predicted to come within an allowed perimeter.

2.2 Cluster Level Command and Control

Depending on the CONOPS, the control of the Cluster may be performed either centrally from *NaFCon*, by a 'leader node' in the cluster controlling the states of the 'cluster team', or fully distributed. A team sports analogy of the centralized control is American football, with its reliance on real-time communication infrastructure. In contrast, team sports such as basketball and soccer rely on each individual player making decisions based on his own situational awareness, strategy and tactics.

In the MIT *Nested Autonomy Prototype* the fully distributed option has been chosen, in large part because of the high level of autonomy made available by the `IvP Helm` , and the latent and intermittent undersea communication infrastructure.

An example of the distributed cluster control is the *Synchronized Swimming* activated in the `TRAIL` sub-state. Here the trailing vehicle will use the `BHV_Trail` behavior to maintain a fixed relative or absolute bearing, and range to a collaborating node, simply by monitoring the status reports it broadcasts. Because of the latency and intermittency, the extrapolation feature of the trail behavior is crucial to the performance. Another important feature is a limit on the validity of the messages received from the other vehicle. Thus, if the collaborating node has not been heard from for a

#	Module Name	Module Description	Author	Size
1	**pNaFCon**	Manages translation of commands from field control to autonomy state, and conversely translates autonomy state to status reports. Manages "back-seat driver" control *Libraries: mbutil, MOOS, MOOSGen, MOOSUtility*	Balasuriya	
2	**pMessageSim**	Used to make initial deploy of vehicle in cases where acoustic communication is not applicable on surface. Optionally makes prosecute command as well. *Libraries: mbutil, MOOS, MOOSGen, MOOSUtility*	Eickstedt	
3	**pSearch**	State transition manager for acoustic source tracking CONOPS. Responsible for managing the transition into the *Prosecute State* in response to a *Prosecute Command*. Re-deploys if target not interceptable. *Libraries: MOOS, MOOSGen, MOOSUtility*	Eickstedt	
4	**pTrackQuality**	State transition manager for acoustic source tracking CONOPS. Responsible for terminating the *Prosecute State*. Also issues redeploy before vehicle reaches operational limits for all states. *Libraries: MOOS, MOOSGen, MOOSUtility*	Schmidt	
5	**pTargetOpportunity**	State transition manager for acoustic source tracking CONOPS. Transitions autonomously from *Deploy* to *Prosecute State*. Monitors incoming *Track reprorts* and issues *Prosecute Command* to ownship if target can be intercepted *Libraries: MOOS, MOOSGen, MOOSUtility*	Schmidt	
6	**pClusterPriority**	This process weights the priority to track a target (using BHV_Attractor) by how close the vehicle is to the target relative to all the collaborating nodes. Closer vehicles are given higher weights, thus leading to a "zone defense" style of multi-vehicle autonomy. The last piece to this "zone defense" scheme is the BHV_RubberBand, which attracts vehicles back to their initial deploy location which increasing strength with distance. *Libraries: tes_util, mbutil, MOOS, MOOSGen, MOOSUtility*	Schneider	
7	**pHuxley**	Does something *Libraries: mbutil, MOOS, MOOSGen, MOOSUtility*	Balasuriya	
8	**pGeneralCodec**	Highly extensible encoder / decoder of messages from human readable integers, booleans, strings, or floats to hexidecimal strings suitable for sending by the WHOI MicroModem. *Libraries:tes_util, MOOS, MOOSGen, MOOSUtility*	Schneider	
9	**pAcommsHandler**	Manages queues for transmission of acoustic messages. Different message queues can be given priorities that increase with time since the last message was sent from that queue. *Libraries: tes_util, MOOS, MOOSGen, MOOSUtility*	Schneider	
10	**pAcommsPoller**	Manages and schedules polling of other nodes on the modem network. *Libraries: tes_util, MOOS, MOOSGen, MOOSUtility*	Schneider	
11	**iMicroModem**	Driver process for WHOI micromodems. *Libraries: tes_util, MOOS, MOOSGen, MOOSUtility*	Grund	
Total unique lines of code Total aggregate lines of code				

Table 1: Autonomous Communication, Command and Control Modules for MOOS-ivP onboard autonomy system.

specified time, the extrapolation of its position will be ignored, and the trailing node will enter a local loiter behavior until a fresh status report from the other vehicle is received, and the trailing resumed.

Another distributed cluster control feature of the prototype is the collaborative search and tracking behaviors activated in response to a *Prosecute Command*. Again these behaviors are carried out without any need for two-way *handshake* communication with the other cluster nodes. The paradigm philosophy is that any useful information from other nodes is fused into the situational awareness, but the individual nodes are not dependent on any outside information for pursuing the mission objective. A characteristic example is the dynamic, distributed assignment of priorities of the cluster behaviors `BHV_Attractor` and `BHV_RubberBand`, with the first attempting to attract the node towards the cued target position and the other trying to maintain the node near the assigned station point. In the sports analogy, the command and control process `pClusterPriority` implements an autonomous *zone defense* strategy by dynamically assigning priorities based on the known position of the other cluster assets and the cued target position. Without requiring any explicit coordination and communication with the other nodes, each node will therefore on its own assign priority to the two competing behaviors. As a result, the cluster node closest to the target will most aggressively initiate pursuit. On the other hand, the other nodes are not sitting idle, but pursuing the target with some finite priority, thus ensuring that they are in a position to more actively initiate the search in case the closest node fails to detect the target or never reacted because it did not receive the cuing message. Again, the soccer analogy is obvious, with all defenders moving in position to react if a forward from the other team is passing by the first line of defense. The actual behaviors associated with this distributed 'zone defense' is never commanded directly, but built into the relative priorities of the competing behaviors allocated by `pClusterPriority`. Based on a large number of virtual and field experiments a range-exponential priority allocation appears optimal.

In addition to the cluster command and control components of the on-board processing also adopts the distributed decision paradigm. For example, the fusion of bearing tracks received from other platforms are applied on collaborating nodes if deemed useful, e.g. for producing a geo-referenced target track (`pMBTracker`), but the nodes are not dependent on it, and will continue to attempt a single node tracking sequence (`p1HTracker`) if it does not receive useful information from other nodes.

2.3 Field Level Command and Control

Although a large amount of autonomous command and control is delegated to the nodes the field level operators obviously have the ultimate control authority with the power to change the roles of the various clusters and nodes under its command. However, in the *Nested Autonomy* paradigm this authority can only be exercised through simple commands that trigger a state transition on the network nodes. The ultimate action associated with the issued commands is entirely defined in the `MOOS-IvP` configurations on the nodes.

The commands for the network nodes are encoded into *Compact Control Language* (CCL) messages which are then transmitted via the acoustic modem network infrastructure. In the *PLUSNet* prototype a limited message set was hard coded into a couple of dedicated CCL messages. This message set was very restrictive, allowing only a handful of state transitions. Thus, to enable a certain *Deploy* or *Prosecute* mode in was necessary to change the configuration files before the

vehicle was deployed. To allow an arbitrarily large state space to be activated without recovery and re-configuration, an new message handler, coder and decoder stack was developed in 2008 by Toby Schneider at MIT. Most of the stack was tested successfully in the August 2008 GLINT'08 experiment. The new software stack was completed in the fall of 2008 with the development of the completely generic coder-decoder (CoDec) `pGeneralCodec`. it uses a dedicated CCL message, assigned number 32, with a highly flexible message format that allows for optimal compression and content, specified in a configuration file which may be changed to optimally fit the specific geographical region and mission suite. The configuration files for the MIT *Nested Autonomy Prototype* are descried in Appendix A.

In an actual operating system the command and control CCL messages may be generated either by the operators directly, or by a topside command and control autonomy system, e.g. linked to an environmental forecasting framework. In the MIT prototype the commands are issued by the operator, typically the Chief Scientist or a trained pilot using the topside MATLAB GUI, `NaFConSim.m`. In combination with the new CCL software stack, it allows the operators to a activate any of the autonomy states without reconfiguration.

The `NaFConSim` GUI is currently configured for issuing *Deploy Commands* with 7 allowed *Mission Types*:

0 Transitions the node to the `SENSING` state, in the current configuration a hexagonal loiter with the on-board sensing and processing active.

1 Transition to `LOWPOWER` state. In the prototype a hexagonal loiter around the deploy location.

2 Transition to `OFF` state. Currently the vehicle transits to the deploy location and surfaces.

3 Transition to `RETURN` state. In the prototype identical to the `OFF` state.

4 Transition to `RACETRACK` state. In prototype configuration the vehicle will run a zero-width racetrack at the specified heading and length, starting at the deploy location.

5 Transition to `YOYO` state. In prototype the vehicle will run the same racetrack as for mission 4, but executing a vertical YoYo survey using the `BHV_SmartYoyo` behavior.

6 Transition to `ZIGZAG` state.

Similarly, the GUI is configured for selecting 4 *Prosecute Mission Types*. All the mission types initially transitions to the `SEARCH` state, followed by the `AMBIGUOUS` state once the cued target event is detected. Once the ambiguities are resolved. the node will transition into one of the `TRACKING` sub-states, depending on the mission type,

1. Transition to `ADAPTIVE` tracking state, where the vehicle will adaptively maneuver to optimize the tracking performance.

2. Same as 1, but once tracking is deemed satisfactory by the `pTrackQuality` process, it will enter the `CLASSIFYING` state, which in the prototype will maintain a heading pointing towards the computed target track.

3. Transition to `TMA` state, where the vehicle in the prototype will enter a zig-zag survey along a specified heading.

4. Transition to a hexagonal loiter similar to the one used in the `SENSING` *Deploy Mission*.

As mentioned earlier, the number of *Deploy* and *Prosecute* commands the communications stack can handle is unlimited. The current suite is dictated only by the capabilities built into the GUI. Thus for example, the `TRAIL` state used for *Synchronized Swimming* can currently only be activated by changing the configuration, e.g. by replacing one of the other deployment modes. This and other useful transitions will be enabled in the next release of the topside command GUI.

3 Communication, Command and Control Modules

3.1 pNaFCon

3.1.1 Brief Overview

pNaFCon is the MOOS process that provides the connectivity between the network communication infrastructure and the autonomy states of the vehicle. As such it is the principal link between the communication infrastructure and the pHelmIvP behavior-based autonomy. Thus, it subscribes to the command messages from the topside and status and contact reports from other nodes and performs the associated state transitions by posting a set of state variables. Thus for example, in response to a topside Deploy Command the MOOS variables DEPLOY_STATE is published by pNaFCon to place the vehicle in its Deploy state, with DEPLOY_MISSION defining the deploy sub-states, such as a low-power drift, a loiter pattern, or an environmental sampling mission with a YoYo in depth along a horizontal racetrack. Similarly, pNaFCon publishes the variables PROSECUTE_STATE and PROSECUTE_MISSION to transition the vehicle autonomy into a particular Prosecute sub-state, such as adaptive target tracking or a traditional TMA ZigZag survey, for example.

NaFCon also is responsible for generating Status and Contact reports, depending on the state of the vehicle. Thus, for example it will respond to a target bearing being published by the real-time data processing by creating a contact Report message, which will then be queued, scheduled and transmitted bythe acoustic communication stack.

3.1.2 Parameters for the pNaFCon Configuration Block

The following configuration parameters are defined for pNaFCon.

ProsecuteMission: Default rosecute mission: 1: Adaptive DLT; 2: Adaptive DLT with subsequent depth-discremination clasification. This parameter used to be necessary since the pFramer CoDec used until recently did not allow more than a single type of Prosecute mission to be commanded via the acoustic link. When using the new generic CoDec pGeneralCoDec this parameter is obsolete and will be eliminated in future versions.

etc etc

Below is an example configuration block for pNaFCon,

Listing 3.1 - An example pNaFCon *configuration block .*

```
1 /////////// Global Variables ////////////////
2 LatOrigin = 36.87
3 LongOrigin = -122.42
4 MagneticOffset = 0.0
5 ////////Configuration Block for pNaFCon ///////////////
6 ProcessConfig = pNaFCon
7 {
8   AppTick   = 4
9   CommsTick = 4
10  ProsecuteMission = 1
11  DestinationPlatformId = 3
12  CollaboratingPlatformId = 4
13  TrackId = 31
14  Orbit_radius =300
15  Orbit_points = 6
```

```
16   ZigZag_Period = 300
17   ZigZag_Amplitude = 150
18   Report_Delay = 2;
19 }
```

The `LatOrigin` and `LonOrigin` parameters on lines 1-2 are not specific to `pNaFCon`, but are specified at the global level in a MOOS file. They are needed to allow the conversion between local x-y coordinates and latitude-longitude coordinates. They are mandatory parameters and `pNaFCon` will refuse to launch if they are lacking - but they are mandatory for other common MOOS applications as well.

3.1.3 MOOS variables subscribed to by pNaFcon:

.

MOOS variable	Type	Description	Published by
NAV_X	D	Local UTM x-coordinate for ownship	pHuxley
NAV_Y	D	Local UTM y-coordinate for ownship	pHuxley
NAFCON_MESSAGES	S	Incoming acomm messages	

3.1.4 MOOS variables published by pNaFCon:

.

MOOS variable	Type	Description	Format
DEPLOY_STATE	$	Defines DEPLOY state.. “FALSE”: Vehicle not in DEPLOY state “OFF”: Vehicle is idleing “DEPLOY”: Vehicle in DEPLOY state “ABORT”: Vehicle transiting to abort point	
MICROMODEM_COMMAND	$	Polling a Collaborating node	
POLL_REQUEST	D	Poll request flag	
COLLABORATOR_ID	$	Collaborating id from moos file	
SINCE_POLLED	D	Time since last poll	
BEARING_STAT	D	Initialize BEARING_STAT = 0	
REMOTE_COLLABORATION	D	State of the collaborating node ”OFF”: No collaboration ”SYNCH”: In synch ”COLLABORATING”: In collaborating mode	
COMMUNITY_STAT	$	Status of the collaborating vehicle	
BEARING_STAT2	$	Contact info. of the collaborating vehicle	
TRACK_STAT2	$	Track info. of the collaborating vehicle	
GOTO_SURFACE	$	Goto surface flag ”TRUE” or ”FALSE”	
DEPLOY_STATION	$	Station keeping position	
TARGET_ID	$	Track number for prosecuted target	5
TARGET_STATE	$	Cued target location and source characteristics for use by target simulators	tgt_x=2000, tgt_y=1525, ... tgt_id=5
TGT_id_NAV_X	$	Cued UTM x coordinate for id’th target	
TGT_id_NAV_Y	$	Cued UTM y coordinate for id’th target	
TGT_id_NAV_HEADING	$	Cued Heading of id’th target	
TGT_id_NAV_SPEED	$	Cued speed id’th targets	
TGT_id_NAV_UTC	$	UTC time of cued target info	
MY_TARGET	$	Current target of interest	TGT_5
CURRENT_TARGET	$	Current target id	
PLUSNET_MESSAGES	$	Status and contact messages to be broadcast by modem	

3.2 pMessageSim

3.2.1 Brief Overview

This MOOS module provides an automated procedure for launching a vehicle into an initial *Deploy* state at some specified delay following launch. It does so by simply posting a `NAFCON_MESSAGES` in the MOOSDB in the same way as `pNaFCon`. It is particularly useful in cases where the conditions do not support the launching via the acoustic modem. In fact it was originally written by D. Eickstedt at sea in the MB06 trial in Montery Bay, where the communication stack was not yet robust enough to provide reliable cammanding of the vehicles. `pMessageSim` also provides the optional launching of a delayed *Prosecute* command to the vehicle itself. After the acomms stack has become further developed, this option is rarely used.

3.2.2 Parameters for the pMessageSim Configuration Block

The following configuration parameters are defined for `pNaFCon`.

`message_type`: *Deploy* mission type. Action depends on settings in behavior configuration file. Current mission suite is

- 0 Continuous DCL hex-loiter deployment: Sensors and processing turned on
- 1 Low-power deployment.
- 2 Off. Go to surface
- 3 Return to station and surface
- 4 Environmental mission. Racetrack
- 5 Evironmental mission. Adaptive vertical yoyo. Horizontal racetrack
- 6 Environmental mission. Horizontal zigzag, constant depth.

`trigger_depth`: Depth at which delay timer for deploy message is triggered. If negative triggering will occur at start-up..

`deploy_delay`: Delay in seconds of deploy message. If not specified, default is 30 seconds.

`prosecute_delay`: Delay in seconds of prosecute message. In most cases set to value larger than mission time to disable autonmatic prosecute message.

`deploy_x`: UTM x-coordinate of deployment point.

`deploy_y`: UTM y-coordinate of deployment point.

`deploy_depth`: Depth of deployment.

`dest_addr`: Destination platform ID. Here Ownship vehicle ID.

`abort_x`: UTM x-coordinate of abort point. When aborting the vehicle will transit to this point at depth and then surface..

`abort_y`: UTM y-coordinate of abort point.

`abort_depth`: Depth at which vehicle will transit to abort point.

`op_radius`: Operations radius allowed for vehicle relative to current deploy location. If deploy command issued to a location outside this range, the state manager process, currently `pTrackQuality`, will issue a local deploy.

`deploy_duration`: Duration in seconds of deploy state. If exceeded without redeploy, vehicle will enter `ABORT` state and transit to abort location and surface.

`target_x`: UTM x-coordinate of target to be prosecuted after `PROSECUTE_DELAY`.

`target_y`: UTM y-coordinate of target to be prosecuted.

`target_depth`: Depth of target to be prosecuted.

`target_heading`: Heading of target to be prosecuted.

`target_speed`: Speed of target to be prosecuted.

`target_id`: Identification number allocated to target to be prosecuted.

`prosecute_duration`: Duration in seconds of prosecute state. If exceeded without redeploy, vehicle will enter `ABORT` state and transit to abort location and surface.

`yoyo_distance`: Length of racetrack legs for *Deploy Mission* 4 and 5, and length of zigzag projection on average heading for *Deploy Mission* 6.

`yoyo_heading`: Heading of racetrack legs for *Deploy Mission* 4 and 5, and average heading for *Deploy Mission* 6.

`yoyo_upper`: Upper turning depth of vertical yoyo.

`yoyo_lower`: Lower turning depth of vertical yoyo.

Below is an example configuration block for`pMessageSim`,

Listing 3.2 - An example `pMessageSim` *configuration block .*

```
 1 ////////////// Global Variables ////////////////
 2 LatOrigin = 36.87
 3 LongOrigin = -122.42
 4 MagneticOffset = 0.0
 5 ////////Configuration Block for pMessageSim ///////////////
 6 ProcessConfig = pMessageSim
 7 {
 8  AppTick    = 4
 9  CommsTick  = 4
10
11  message_type = 1
12  trigger_depth = -1
```

```
13  prosecute_delay = 60000
14  deploy_x = 5000 // 500 A6 racetrack
15  deploy_y = 9000 // 2100 A6 racetrack
16  deploy_depth = 25.0
17  dest_addr = 3
18  abort_x = 4500
19  abort_y = 8500
20  abort_depth = 6.0
21  op_radius = 5000
22  deploy_duration = 7200
23  target_x = 0
24  target_y = 1000
25  target_depth = 20.0
26  target_heading = 90.0
27  target_speed = 2.0
28  target_id = 5
29  prosecute_duration = 1800
30  yoyo_distance = 4000
31  yoyo_heading = 135
32  yoyo_upper = 5
33  yoyo_lower = 30
34 }
```

The `LatOrigin` and `LonOrigin` parameters on lines 1-2 are not specific to `pMessageSim`, but are specified at the global level in a MOOS file. They are needed to allow the conversion between local x-y coordinates and latitude-longitude coordinates. They are mandatory parameters and `pMessageSim` will refuse to launch if they are lacking - but they are mandatory for other common MOOS applications as well.

3.2.3 MOOS variables subscribed to by pMessageSim:

.

MOOS variable	Type	Description	Published by
NAV_DEPTH	D	Current ownship depth. Used fro triggering message	pHuxley

3.2.4 MOOS variables published by pMessageSim:

.

MOOS variable	Type	Description	Format
NAFCON_MESSAGES	$	Deploy or prosecute message for ownship.	

3.3 pSearch

3.3.1 Brief Overview

pSearch is a transition manager for the transition from *Deploy State* to *Prosecute State* in response to a *Prosecute Command* issued either by **NaFCon** , pMessageSim or the target watchdog process pTargetOpportunity described later.

Once a NAFCON_MESSAGES command is reived from the MOOSDB, pSearch will first detemine whether the target is interceptable within the specified distance. If not, it will return the vehicle to its earlier *Deploy State*. If the target is interceptable it will continuously monitor the target and ownship motion, checking for continued interceptability. If the minimum distance to the expected target position is reached with detection, *pSearch* will enter a local loiter pattern iwth the sensors remaing active for possible detection of the target event.

3.3.2 Parameters for the pSearch Configuration Block

min_range: Range from predicted target position at which vehicle will enter waiting loiter pattern until possibly detecting the target event.

orbit_radius: Radius applied in waiting loiter hexagon awaiting possible event detection

giveup_range: Target range in meters beyond which the prosecute will not be pursued. Vehicle will remain in or return to *Deploy State*

Below is an example configuration block forpSearch,

Listing 3.3 - An example pSearch *configuration block .*

```
 1 ////////////// Global Variables ////////////////
 2 LatOrigin = 36.87
 3 LongOrigin = -122.42
 4 MagneticOffset = 0.0
 5 ////////Configuration Block for pSearch ///////////////
 6 ProcessConfig = pSearch
 7 {
 8   AppTick   = 4
 9   CommsTick = 4
10
11   min_range = 500
12   orbit_radius = 300
13   giveup_range = 2500
14 }
```

3.3.3 MOOS variables subscribed to by pSearch:

.

MOOS variable	Type	Description	Published by
VEHICLE_ID	D	Node number for ownship.	pNaFCon
NAV_X	D	Ownship UTM x-coordinate	pHuxley
NAV_Y	D	Ownship UTM y-coordinate	pHuxley
PROSECUTE_STATE	$	Prosecute state	pNaFCon
TARGET_STATE	$	Assumed target state vector from *Prosecute Command*	pNaFCon
SENSOR_DEPLOY	$	Current *Deploy* location	pNaFCon
SENSOR_DEPTH_DEPLOY	$	Current *Deploy* depth	pNaFCon
DEPLOY_RADIUS	D	Maximum operating radius in meters	pNaFCon
ABORT_LAT	D	Abort location latitude in decimal degrees	pNaFCon
ABORT_LONG	D	Abort location longitude in decimal degrees	pNaFCon
ABORT_TIME	D	UTC Abort time	pNaFCon
SENSOR_DEPTH_ABORT	D	Depth ofr transit to abort location	pNaFCon
TRACKING	$	Tracking state. Indicates whether target has been detected or not.. "NO_TRACK": No detection yet "AMBIGUOUS": Detection, but ambiguous bearing "TRACKING": Un-ambiguous bearing tracking "CLASSIFYING": Classifying sub-state	pBearingTrack

3.3.4 MOOS variables published by pSearch:

.

MOOS variable	Type	Description	Format
NAFCON_MESSAGES	$	Deploy for ownship published if the intercept conditions for a reported target are satisfied.	
. TRACK_STAT	$	Comma separated track state vector. Initializes to state=0. If target not detected the target state variable is set to state=4.	node=3,state=0,x=...
IN_RANGE	$	Flag indicating whether vehicle is within minimum range to expected target position.	TRUE FALSE

3.4 pTrackQuality

3.4.1 Brief Overview

This is the principal autonomus state transition managing process responsible for terminating the *Prosecute State* for acoustic source tracking missions. It is monitoring a suite of MOOS variables and performs state transitions accordingly. It's main objective is to terminate a source tracking *Prosecute Mission*, and return the vehicle to its default *Deploy State*. The transition is triggered either by the tracking solution with satisfactory uncertainty being achieved, or the maximum number of *Track Reports* is reached.

`pTrackQuality` is also monitoring the distance to the boundaries of the operations box and the distance from the station point, and will issue a redeploy if too close, to avoid the *Surfacing* triggered by the helm if the boundary is exceeded. This feature is active in both the *Deploy* and *Prosecute* states.

3.4.2 Parameters for the pTrackQuality Configuration Block

`bearing_rate_threshold:` If positive sets the threshold for terminating source tracking. When bearing rate decreases below the set percentage of the max imum bearing rate. Can produce undesired terminations if set too high. Rarely used.

`track_limit:` Maximum duration of *Tracking Sub-State* in seconds. When exceeded, vehicle will be re-deployed.

`min_opbox_distance:` Minimum allowed distance to operations box boundary. Triggers re-deploy when reached to avoid 'hard' abort by Helm.

`max_classify_time:` Maximum duration of *Classify Sub-State* of source tracking prosecute in seconds. When exceeded, vehicle will be re-deployed.

`track_sigma_single:` Standard deviation threshold for single vehicle track solution in meters. `TRACK_REPORT` will be issued when computed standard deviation is less than this value.

`buffer_length_single:` Number of single vehicle track solutions used in computing track uncertainty.

`track_sigma_multiple:` Standard deviation threshold for multi-vehicle, collaborative track solution in meters. `TRACK_REPORT` will be issued when computed standard deviation is less than this value.

`buffer_length_multiple:` Number of multi-vehicle, collaborative track solutions used in computing track uncertainty.

`track_report_time:` Duration of time period where *Track Reports* are issued. When exceeded, vehicle will transition to *Deploy State*.

`deploy_location:` Location of re-deploy. Can be ither 'station' or 'present', for return to station point or preset position, respectively.

Below is an example configuration block for`pTrackQuality`,

Listing 3.4 - An example `pTrackQuality` *configuration block .*

```
 1 ////////////// Global Variables ////////////////
 2 LatOrigin = 36.87
 3 LongOrigin = -122.42
 4 MagneticOffset = 0.0
 5 ////////Configuration Block for pTrackQuality ///////////////
 6 ProcessConfig = pTrackQuality
 7 {
 8   AppTick = 1
 9   CommsTick = 1

10   bearing_rate_threshold = 0  // percent of max achieved during tracking
11   track_limit = 600           // Tracking duration in seconds
12   min_opbox_distance = 500    // min distance to opreg perimeter
13   max_classify_time  = 600    // max Classify time for PROSECUTE_MISSION = 2
14   track_sigma_single = 100
15   buffer_length_single = 10
16   track_sigma_multiple = 100
17   buffer_length_multiple = 3
18   track_report_time = 180
19   deploy_location = station   // Options: station or present
20 }
```

The `LatOrigin` and `LonOrigin` parameters on lines 1-2 are not specific to `pMessageSim`, but are specified at the global level in a MOOS file. They are needed to allow the conversion between local x-y coordinates and latitude-longitude coordinates. They are mandatory parameters and `pMessageSim` will refuse to launch if they are lacking - but they are mandatory for other common MOOS applications as well.

3.4.3 MOOS variables subscribed to by pTrackQuality:

.

MOOS variable	Type	Description	Published by
VEHICLE_ID	D	Node number for ownship.	pNaFCon
TGT_#_NAV_X	D	UTM x-coordinate of target. Initially from *Prosecute Command*. Updated by tracking solution	pNaFCon p1HTracker pMBTracker
TGT_#_NAV_Y	D	UTM x-coordinate of target. Initially from *Prosecute Command*. Updated by tracking solution	pNaFCon p1HTracker pMBTracker
TGT_#_NAV_UTC	D	UTC time of target location	p1HTracker pMBTracker
NAV_X	D	Ownship UTM x-coordinate	pHuxley
NAV_Y	D	Ownship UTM y-coordinate	pHuxley
SENSOR_DEPLOY	$	Current *Deploy* location	pNaFCon
SENSOR_DEPTH_DEPLOY	$	Current *Deploy* depth	pNaFCon
PROSECUTE_STATE	$	Prosecute state	pNaFCon
PROSECUTE_MISSION	$	Prosecute mission type	pNaFCon
BEARING_STAT	$	Comma-separated bearing state variables: vehID,state,bearing,x,y,beamno,sigma,utc	pBearingTrack
TRACK_STAT	$	Tracking state vector	p1HTracker
TRACKING	$	Tracking state. "NO_TRACK": No detection yet "AMBIGUOUS": Detection, but ambiguous bearing "TRACKING": Un-ambiguous bearing tracking "CLASSIFYING": Classifying sub-state	pBearingTrack
TRACKING_MODE	$	Tracking mode: "SINGLE": Single vehicle tracking "MULTIPLE": Multi-vehicle cross-bearing tracking	pMBTRacker
CLOSE_RANGE	$	Reset flag for geo trackers	pSearch
CLASS_DONE	$	Flag set to `TRUE` when classification substate is completed.	pTrackQuality
. DEPLOY_RADIUS	D	Maximum operating radius in meters	pNaFCon
OPREG_TRAJECTORY_PERIM_DIST	D	Current distance to operations box in meters	pHelmIvP
ABORT_LAT	D	Abort location latitude in decimal degrees	pNaFCon
ABORT_LONG	D	Abort location longitude in decimal degrees	pNaFCon
ABORT_TIME	D	UTC Abort time	pNaFCon
SENSOR_DEPTH_ABORT	D	Depth ofr transit to abort location	pNaFCon

3.4.4 MOOS variables published by pTrackQuality:

.

MOOS variable	Type	Description	Format
NAFCON_MESSAGES	$	Deploy for ownship published if any of the prosecute termination conditions are satisfied.	
. TRACK_REPORT	$	*Track Report* issued when uncertainty satisfactory. Translated into modem message by `pNaFCon`	
TRACKING	$	Tracking substate flag. Changed to "CLASSIFYING" if `PROSECUTE_MISSION = 2` and tracking is satisfactory	
CLASS_DONE	$	Flag set to `TRUE` when classification substate is completed.	
.			

3.5 pTargetOpportunity

3.5.1 Brief Overview

`pTargetOpportunity` is a transition manager that autonomously transitions from *Deploy State* to *Prosecute State* if a *Track Report* is received from a collaborating vehicle for an event, such as an acoustic source or a plume, is heading in a direction that allows ownship to intercept. It currently represents the highest level of autonomous decisionmaking on the vehicle.

When a *Track report* is received from the collaborting vehicle the closest point of approach is calculated, and the prosecute command will be issued only if the CPA is les than a maximum range specified in the configuration file.

There are currently one of two algorithms apllied, dependent on the compiler settings. The ultimate one uses the standard MOOS-IvP CPA engine, but this algorithm needs debugging. The other uses a simple minimum distance computation ignoring the estimated target speed. At this point both set of configurationj parameters are specified.

3.5.2 Parameters for the pTargetOpportunity Configuration Block

`CollaboratorNodes:` Comm-separated list of vehicle ID for trusted collaborator nodes. Only *Track Reports* from nodes on the list will be acted upon.

`OpportunityMission:` *Prosecute Mission* to be applied:

1. Adaptive tracking mission
2. Adaptive tracking with subsequent classification sub-state
3. Zig-Zag mission at estimated target heading
4. Loiter deploy pattern ith tracking.

`OpportunityRadius:` Minimum target distance, below which a *Prosecute Command* is issued to ownship.

`VehicleMaxSpeed:` Maximum estimated speed of target to be intercepted

`ThresholdCPA:` Threshold range for CPA below which a prosecute sequence will be initiated

`ThresholdTOL:` Tolerance of the CPA determination.

Below is an example configuration block for`pTargetOpportunity`,

Listing 3.5 - An example `pTargetOpportunity` *configuration block .*

```
1 /////////// Global Variables ////////////////
2 LatOrigin = 36.87
3 LongOrigin = -122.42
4 MagneticOffset = 0.0
5 ////////Configuration Block for pTargetOpportunity ///////////////
6 ProcessConfig = pTargetOpportunity
7 {
8   AppTick = 4
9   CommsTick = 4
```

```
10
11   CollaboratorNodes = 0,4
12   OpportunityMission = 1
13   OpportunityRadius = 2000
14   VehicleMaxSpeed = 2.0
15   ThresholdCPA = 1000
16   ThresholdTOL = 20
17 }
```

The `LatOrigin` and `LonOrigin` parameters on lines 1-2 are not specific to `pMessageSim`, but are specified at the global level in a MOOS file. They are needed to allow the conversion between local x-y coordinates and latitude-longitude coordinates. They are mandatory parameters and `pMessageSim` will refuse to launch if they are lacking - but they are mandatory for other common MOOS applications as well.

3.5.3 MOOS variables subscribed to by pTargetOpportunity:

.

MOOS variable	Type	Description	Published by
VEHICLE_ID	D	Node number for ownship.	pNaFCon
COLLABORATOR_ID	D	Node number for collaborating vehicle	pNaFCon
NAV_X	D	Ownship UTM x-coordinate	pHuxley
NAV_Y	D	Ownship UTM y-coordinate	pHuxley
NAV_DEPTH	D	Current ownship depth in meters	pHuxley
DEPLOY_STATE	$	Deploy state	pNaFCon
PROSECUTE_STATE	$	Prosecute state	pNaFCon
TRACK_STAT2	$	Tracking state vector received from collaborating vehicle	pNaFCon
DEPLOY_RADIUS	D	Maximum operating radius in meters	pNaFCon
OPREG_TRAJECTORY_PERIM_DIST	D	Current distance to operations box in meters	pHelmIvP
ABORT_LAT	D	Abort location latitude in decimal degrees	pNaFCon
ABORT_LONG	D	Abort location longitude in decimal degrees	pNaFCon
ABORT_TIME	D	UTC Abort time	pNaFCon
SENSOR_DEPTH_ABORT	D	Depth ofr transit to abort location	pNaFCon

3.5.4 MOOS variables published by pTargetOpportunity:

.

MOOS variable	Type	Description	Format
NAFCON_MESSAGES	$	Deploy for ownship published if the intercept conditions for a reported target are satisfied.	
.			

3.6 pClusterPriority

3.6.1 Brief Overview

This MOOS module does a darn good job at doing what it is supposed to do

3.6.2 Parameters for the pClusterPriority Configuration Block

The following configuration parameters are defined for `pClusterPriority`.

`verbose:` Level of diagnostic verbosity during runtime. Choises are `verbose`, `terse`, and `quiet`.

`target:` Name of target prosecuted by the cluster. Can be fixed, e.g. `TGT_6` or `BOBBY`. If specified to 'target_updates' the target identification is dynamically allocated. The MOOS variable containing the dynamical target ID is specified in the configuration parameter `target_updates`.

`target_updates:` Identifies the MOOS variable containiing the ID for the target currently being prosecuted. If not specified default is `TARGET_ID`.

`behavior_updates:` Identifies the MOOS variable used for publishing the behavior updates, e.g. for use by `BHV_Attractor`. If not specified default is `MY_TARGET`.

`ownship:` Node ID for ownship, case insensitive, e.g. `UNICORN`.

`friend:` Node ID for collaborating cluster node, case insensitive, e.g. `CARIBOU`. Must be specified for each cllaborating node.

`pwt:` Base priority weight for attracting behavior, e.g. `BHV_Attractor`.

`max_delay_friends:` Maximum allowed delay in seconds for status reports recveived from collaborating nodes. If exceeded the node will be ignored in computing attractor priorities.

Below is an example configuration block for `pClusterpriority`,

Listing 3.6 - Sample `pClusterPriority` *configuration block .*

```
 1 ProcessConfig = pClusterPriority
 2 {
 3    AppTick    = 4
 4    CommsTick  = 4
 5    verbosity = verbose
 6    target = target_updates
 7    target_updates = TARGET_ID
 8    behavior_updates = MY_TARGET
 8    ownship = unicorn
 9    friend = macrura
10    friend = OEX
11    pwt =  100
12    max_delay_friends= 60
13 }
```

3.6.3 MOOS variables subscribed to by pClusterPriority:

.

3.6.4 MOOS variables published by pClusterPriority:

.

3.7 pHuxley

3.7.1 Brief Overview

`pHuxley` is the process which interface with the Main Vehicle Computer (MVC-Huxley) of the Bluefin AUV. During start up pHuxley requests the data streams required from the MVC. Once the back-seat driver is turned on (i.e. when the front-seat driver (MVC) is ready to receive commands from the back-seat) a flag is set and pHuxley starts sending desired speed, heading and depth values to the MVC. pHuxley controls the hand-offs between the front-seat and the back-seat.

3.7.2 Parameters for the pHuxley Configuration Block

3.7.3 MOOS variables subscribed to by pHuxley:

.

3.7.4 MOOS variables published by pHuxley:

.

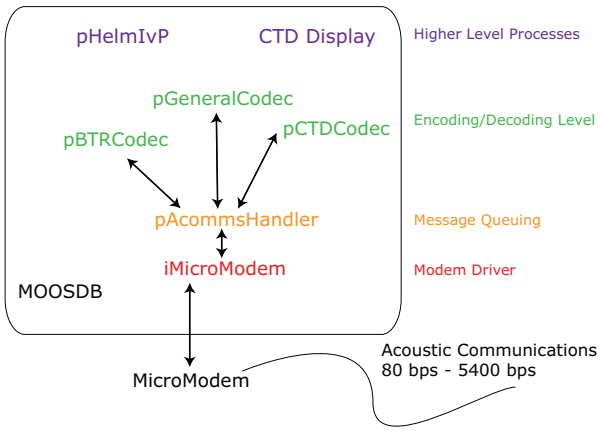

Figure 2: schematic of current acoustic modem stack

3.8 pGeneralCodec

3.8.1 Brief Overview

Any terms in *italics* are defined the Glossary (section 3.8.9).

Problem Communication between multiple autonomous vehicles is often subject to severe throughput limitations, especially in the ocean environment, where highly rate-limited acoustic channels are often the only option. In order to make the best use of this available bandwidth, messages need to be compacted to a minimal size before sending.

Background Figure 2 illustrates the current status of the acoustic communication software "stack" from the viewpoint of this laboratory. pGeneralCodec fits in the level of message encoding/decoding, as suggested by the name.

Solution pGeneralCodec is an process that proposes to be a practical first-pass solution to the difficult problem posed here. It reduces the data required to be sent by:

- Predefined messages: the user must specify a message structure what specifies what fields the message contains and how large each field should be (in an intuitive fashion that pGeneralCodec turns into bits). Both the sender and receiver have preshared knowledge of the message structure. From this knowledge, no meta information about the message (beyond an identifier) needs to be sent, simply the data.
- Custom field sizes: message fields are defined with custom tolerances (ranges and precisions) that are tighter than those given by the IEEE standards for floating point and integer numbers. For example, if a field needs to hold an integer that will never range outside [0, 1000] that field in the message will only be 10 bits long (ceil($\log_2 1001$)).

Furthermore, pGeneralCodec has powerful parsing abilities for both encoding and decoding, including the ability to perform certain geodesic conversions (latitude, longitude $\leftrightarrow$ UTM x,y) and lookups (*modem id* $\leftrightarrow$ vehicle name) on data.

3.8.2 Parameters for the pGeneralCodec Configuration Block

Example .moos file The moos file is simple since the bulk of the configuration is stored in separate XML files (see section 3.8.5 for the configuration of these files):

```
//---------------------------------------------------------------
// pGeneralCodec configuration  block

ProcessConfig = pGeneralCodec
{
  // available to all moos processes.
  AppTick    = 4
  CommsTick  = 4

  // available to all tes moos processes

  //verbose, terse, quiet
  verbosity = verbose

  message_file = example1.xml
  message_file = example2.xml
  message_file = bobs_special_command_set.xml

}
```

Filling out the .moos file

- `verbosity`: choose verbose for full text terminal output, terse for symbolic heartbeat output, and quiet for no terminal output.
- `message_file`: path to an XML file containing a message set of one or messages.

3.8.3 MOOS variables subscribed to by pGeneralCodec:

All the variables subscribed to by pGeneralCodec are configured within the message XML files. See section 3.8.5 and beyond for details on filling out and interpreting these XML files. Thus, in the following table instead of MOOS variables, XML tags from the message XML file are listed. For each `<message></message>` defined, a subscription takes place for the MOOS variable specified at run time within the tag. For example, the .moos file includes a file `message_file = example1.XML`, and `example1.XML` contains the tag `<incoming_hex_moos_var>STATUS_HEX_30B</incoming_hex_moos_var>`. Thus, `STATUS_HEX_30B` is subscribed for.

XML tag specifying a MOOS variable	Type	Description	Published by
<incoming_hex_moos_var/>	$	Hexadecimal string to decode (probably from MicroModem).	pAcommsHandler (or MOOSBlink)
<destination_moos_var key="destkey"/>	$ or D	Contains the modem_id to send this message from. Can either be double (ex: 3) or string (ex: ...,destkey=3,...)	pNaFCon (PLUS-NET_MESSAGES) or others
<trigger_moos_var/>	$ or D	A publish here triggers the creation of this message. The contents may contain message parts or not.	pNaFCon (PLUS-NET_MESSAGES) or others
<moos_var key="somekey"/>	$ or D	Data for a given *message_var*. Can either be double (ex: 3.234) or string (ex: "bob" or "3.234") or keyed string (ex: ...,somekey=3.234,...).	pNaFCon (PLUS-NET_MESSAGES) or others

3.8.4 MOOS variables published by pGeneralCodec:

Similarly to the subscriptions, the publishes done by pGeneralCodec are all defined in the message XML files. Again, instead of MOOS variables, the table below indicates the XML tags for which one can define the publishes.

XML tag specifying a MOOS variable	Type	Description	Format
<outgoing_hex_moos_var/>	$	Encoded hexadecimal string (likely to be subscribed for by pAcommsHandler).	if the destination is NOT broadcast (*modem_id=0*: "Dest=*modem_id*, HexData=*2N character hex string where N is bytes defined by <size></size>*", else "*2N character hex string*"
<publish> <moos_var type="string"/> </publish>	$	(string) Message created from decoded hexadecimal string.	Defined by <format></format>
<publish> <moos_var type="double"/> </publish>	D	(double) Message created from decoded hexadecimal string.	Defined by <format></format>

3.8.5 Usage

Compilation pGeneralCodec depends on the boost string library and the xerces-c XML parsing library in addition to the libraries included in MOOS and moos-ivp-local.

- boost: reference *http://www.boost.org/* or look for your distribution's boost developer package (libboost-dev in debian/ubuntu).
- xerces-c: reference *http://xerces.apache.org/xerces-c/* or look for your distribution's xerces-c developer package (libxerces28-dev in debian/ubuntu as of this writing).

Example message XML file Also see section 3.8.7 for further examples. Let's call this file example1.xml:

```
<?xml version="1.0" encoding="ASCII" standalone="yes"?>
<message_set>
  <message>
    <name>GoToCommand</name>
    <outgoing_hex_moos_var>OUT_GOTO_HEX</outgoing_hex_moos_var>
    <incoming_hex_moos_var>IN_GOTO_HEX</incoming_hex_moos_var>
    <destination_moos_var key="Destination">OUTGOING_COMMAND</destination_moos_var>
    <trigger>publish</trigger>
    <trigger_moos_var mandatory_content="CommandType=GoTo">OUTGOING_COMMAND</trigger_moos_var>
    <size>30</size>
    <layout>
      <static>
        <name>type</name>
        <value>goto</value>
      </static>
      <int>
        <name>goto_x</name>
        <max>10000</max>
        <min>0</min>
      </int>
      <int>
        <name>goto_y</name>
        <max>10000</max>
        <min>0</min>
      </int>
      <bool>
        <name>lights_on</name>
      </bool>
      <string>
        <moos_var>SPECIAL_INSTRUCTIONS</moos_var>
        <name>new_instructions</name>
        <max_length>10</max_length>
      </string>
      <float>
        <name>goto_speed</name>
        <max>3</max>
        <min>0</min>
        <precision>2</precision>
      </float>
    </layout>
    <publish>
      <moos_var>INCOMING_COMMAND</moos_var>
      <all />
    </publish>
    <publish>
      <moos_var>SPECIAL_INSTRUCTIONS</moos_var>
      <format>special_instructions=%1%,lights_on=%2%</format>
      <message_var>new_instructions</message_var>
```

```
      <message_var>lights_on</message_var>
    </publish>
  </message>
  <message>
    <name>VehicleStatus</name>
    <trigger>time</trigger>
    <trigger_time>30</trigger_time>
    <outgoing_hex_moos_var>OUT_STATUS_HEX</outgoing_hex_moos_var>
    <incoming_hex_moos_var>IN_STATUS_HEX</incoming_hex_moos_var>
    <size>30</size>
    <layout>
      <float>
        <name>nav_x</name>
        <moos_var>NAV_X</moos_var>
        <max>1000</max>
        <min>0</min>
        <precision>1</precision>
      </float>
      <float>
        <name>nav_y</name>
        <moos_var>NAV_Y</moos_var>
        <max>1000</max>
        <min>0</min>
        <precision>1</precision>
      </float>
      <enum>
        <name>health</name>
        <moos_var>VEHICLE_HEALTH</moos_var>
        <value>good</value>
        <value>low_battery</value>
        <value>abort</value>
      </enum>
    </layout>
    <publish>
      <moos_var>STATUS_SUMMARY</moos_var>
      <all />
    </publish>
  </message>
</message_set>
```

Filling out the message XML file All the messages defined in the XML message files included in the .moos file define the entirety of the available message set. First, a brief background on XML (e**X**tensible **M**arkup **L**anguage). XML files contain tags (like `<name></name>`) that are considered "metadata" and define both the structure of the following data and the contents. Order of the tags does not matter for a given level unless explicitly specified. Text data resides both in the tags (like `<name>bob</name>` or as attributes of the tag (such as `<name id="1245"></name>`). XML files can be edited with any text editor. For more information on XML consult any number of books on the subject or browse the great internet. It is a very widely used format for storing data that can be both read by both people and computers.

Allowed tags Let us first give a description of the allowed tags and then we will go into a description of how to go about making a command file.

- `<?xml version="1.0" encoding="ASCII" standalone="yes"?>`: specifies the file is XML and which schema to use. When schema checking is enabled, this line will change some. All you need to know is that this must be the first line of every message XML file.
- `<message_set></message_set>`: the root element. All XML files must have a single root element. Since we are define a set of messages (one or more per file), this is a logical choice of name for the root element. [mandatory, one allowed].
- `<message></message>`: defines the start of a message. [mandatory, one or more allowed].
  - `<name></name>`: a human readable name for the message. This is not used internally at this point in time. [mandatory, one allowed]
  - `<trigger></trigger>`: how the message is created. Currently this field must take the value "publish" (meaning a message is created on a publish event to a certain moos variable) or "time" (a message is created on a certain time interval). [mandatory, one allowed]
  - `<trigger_moos_var></trigger_moos_var>`: used if `<trigger>publish</trigger>`, this field gives the moos variable that pGeneralCodec should look for publishes to in order to trigger the creation of this message [mandatory iff `<trigger>publish</trigger>`]. optional attribute `mandatory_content` specifies a string that must be a substring of the contents of the trigger variable in order to trigger the creation of a message. For example, if you wanted to create a certain message every time `COMMAND` contained the string `CommandType=GoTo...` but no other time, you would specify `mandatory_content="CommandType=Go` within this tag.
  - `<trigger_time></trigger_time>`: used if `<trigger>time</trigger>`, this field gives the time interval pGeneralCodec should create this message. For example, a value of `<trigger_time>10</trigger_time>` would mean a message was created every ten seconds. [mandatory iff `<trigger>time</trigger>`].
  - `<size></size>`: the size of the message in bytes. There are eight bits (binary digits) to a byte. Use $N - 2$ here for messages passed through pAcommsHandler where N is the desired micromodem frame size (N =32, 64, or 256 depending on the rate). If the `<layout></layout>` of the message exceeds this size, pGeneralCodec will exit on startup with information about sizes, from which you can remove or reduce the size of certain *message_var*s.
  - `<outgoing_hex_moos_var></outgoing_hex_moos_var>`: where to publish the encoded message (as a hexadecimal string). [mandatory, one allowed].
  - `<incoming_hex_moos_var></incoming_hex_moos_var>`: where to look for messages (hex string) to decode. [mandatory, one allowed].
  - `<destination_moos_var></destination_moos_var>`: moos variable to find where this message should be sent. Specify attribute "key=" to specify a substring to look for within the value of this moos variable. For example, if `COMMAND` contained the string

`Destination=3` and you want this message sent to modem id 3, then you should set `key=Destination` to properly parse that string. [optional: default is 0 (broadcast), one allowed].

- `<layout></layout>`: defines the message structure itself (what fields [the message variables or *message_var*s] the message contains and how they are to be encoded). [mandatory, one allowed].
  - `<static></static>`: a *message_var* that is not actually sent with the message but can be used to include in received messages (*publishes*). [optional, one or more allowed].
    - `<name></name>`: the name of this *message_var*. [mandatory, one allowed].
    - `<value></value>`: the value of this static variable. [mandatory, one allowed].
  - `<bool></bool>`: a boolean (true or false) *message_var* [optional, one or more allowed].
    - `<name></name>`
    - `<moos_var></moos_var>`: the moos variable from which to pull the value of this field. [optional if `<trigger>publish</trigger>`: default is trigger_moos_var; mandatory if `<trigger>time</trigger>`, one allowed].
  - `<int></int>`: an integer *message_var* [optional, one or more allowed].
    - `<name></name>`
    - `<moos_var></moos_var>`
    - `<max></max>`: the maximum value this field can take. [mandatory, one allowed].
    - `<min></min>`: the minimum value this field can take. [mandatory, one allowed].
  - `<float></float>`: a floating point *message_var* [optional, one or more allowed].
    - `<name></name>`
    - `<moos_var></moos_var>`
    - `<max></max>`
    - `<min></min>`
    - `<precision></precision>`: an integer that specifies the number of decimal digits to preserve. Negatives are allowed. For example, `<precision>2</precision>` rounds 1042.1234 to 1042.12; `<precision>-1</precision>` rounds 1042.1234 to 1.04e3. [mandatory, one allowed].
  - `<string></string>`: an ASCII string *message_var* [optional, one or more allowed].
    - `<name></name>`
    - `<moos_var></moos_var>`
    - `<max_length></max_length>`: the length of the string value in this field. Longer strings are truncated. `<max_length>4</max_length>` means "ABCDEFG" is sent as "ABCD". [mandatory, one allowed].
  - `<enum></enum>`: an enumeration *message_var* [optional, one or more allowed].
    - `<name></name>`
    - `<moos_var></moos_var>`
    - `<value></value>`: a possible value (string) the enum can take. Any number of values can be specified. [mandatory, one or more allowed].

– `<publish></publish>`: defines a single output value upon receipt of a message. Any number of publishes containing any subset of the *message_var*s can be specified. [mandatory, one or more allowed].

  * `<moos_var></moos_var>`: the name of the moos variable to publish to. If desired, a format string is allowed here as well (e.g. `%1%_NAV_X` will fill `%1%` with the first *message_var*). See the `<format></format>` tag description for more info. [mandatory, one allowed].
  * `<format></format>`: a string conforming to the format string syntax of the boost::format [1] library. This field will specify the format of the string published to the moos variable defined in `<moos_var></moos_var>`. At its simplest it is a string of incrementing numbers surrounded by %%. Or, instead, you may also use a printf style string, using %d for int *message_var*, %lf for floats, and %s for strings, bools and enums. [optional: default is `name1=%1%,name2=%2%,name3=%3%`, where `name1` is the name of the first `<message_var></message_var>` field to follow, `name2` is the second, etc. exception: default is `%1%` if only a single `<message_var></message_var>` defined. one allowed].
  * `<message_var></message_var>`: the name (`<name></name>` above) of a *message_var* contained in this message (i.e. an `<int></int>`, `<bool></bool>`, etc.) the values of these fields upon receipt of a message will be used to populate the format string and the result will be published to `<moos_var></moos_var>`. [mandatory unless `<all />` used, one or more allowed].
  * `<all />`: equivalent to `<message_var></message_var>` for all the *message_var*s in the message. This is a shortcut when you want to publish all the data in a human readable string. [optional, one allowed].

Designing a publish triggered message We will look at two scenarios and detail how to design a proper message file for each scenario. We will reference the example file given in section 3.8.5 for both scenarios.

Scenario: you want to command an surface craft to move to a new location:

1. Identify the data: location (x (`goto_x`) and y (`goto_y`) on a local grid). you also want to specify a speed (`goto_speed`) at which it should transit, whether it should have lights (`lights_on`) on or not, and finally a string (`special_instructions`) with possible special instructions. All these data will come in to a moos variable `OUTGOING_COMMAND` on a string like:

```
OUTGOING_COMMAND: Destination=3,CommandType=GoTo,goto_x=351,goto_y=294,
                  lights_on=true,special_instructions=make_toast,goto_speed=2.3
```

2. Type the data (i.e. is it an int, a float, a string?) and give the ranges and precisions needed:

   - `goto_x`: integer (in meters) (`int`) that will operate on a (positive valued) local grid not to exceed 10 km in either dimension.
   - `goto_y`: same as `goto_x`.

[1] see the syntax of the **format-string** at *http://www.boost.org/doc/libs/1_37_0/libs/format/doc/format.html#syntax*

- `goto_speed`: speed in m/s. the vehicle cannot exceed 3 m/s and does not go backwards. we would like to give precise speeds to the hundredths place. thus, we need a `float` ranging from 0 to 3 with precision 2.
- `lights_on`: simply a flag (boolean value) whether to have our lights on or off. thus, we need a `bool` *message_var*.
- `special_instructions`: We want a field that can hold any string of characters, but we know it will not exceed ten characters. thus, we need a `string` *message_var*.

3. Putting all this together, we can define the `<layout></layout>` portion of the first message defined in section 3.8.5. We do not need any `<moos_var></moos_var>` tags within the *message_var*s since all the data are contained in the contents of the trigger variable message (`OUTGOING_COMMAND`). That is, when we leave out the `<moos_var></moos_var>`, pGeneralCodec will insert `<moos_var>OUTGOING_COMMAND</moos_var>`, which is exactly what we want. For example, taking one of the *message_var*s:

```
<int>
  <name>goto_x</name>
  <max>10000</max>
  <min>0</min>
</int>
```

   is exactly the same as saying

```
<int>
  <name>goto_x</name>
  <moos_var>OUTGOING_COMMAND</moos_var>
  <max>10000</max>
  <min>0</min>
</int>
```

4. Now we can fill out the rest of the tags on the `<message></message>` level:

   - `<name>GoToCommand</name>`: just a name so we can identify this message quickly when reading through the XML.
   - `<outgoing_hex_moos_var>OUT_GOTO_HEX</outgoing_hex_moos_var>`: we will publish our hex strings here for consumption by the low level modem stack processes (probably pAcommsHandler `send=OUT_GOTO_HEX...`). the publishes will look a bit like:

     `OUT_GOTO_HEX: Dest=3,HexData=a9385bc098109830a9385bc0981098a9385bc098109830a9385bc0981098`

   - `<incoming_hex_moos_var>IN_GOTO_HEX</incoming_hex_moos_var>`: where we expect to receive incoming messages to decode (probably from pAcommsHandler `receive=IN_GOTO_HEX...`) messages should be pure hex with the community set to the sending *modem id*. An example for the received message on *modem id*=3 if the sending node (say a topside command computer) is *modem id*=1:

     `IN_GOTO_HEX: a9385bc098109830a9385bc0981098a9385bc098109830a9385bc0981098 {Community=1}`

   - `<destination_moos_var key="Destination">OUTGOING_COMMAND</destination_moos_var>`: we want to pull the destination *modem id* from the same string as the data (moos variable `OUTGOING_COMMAND` and it is found in a key called "Destination".

- `<trigger>publish</trigger>`: we are creating this message on a **publish** (to `OUTGOING_COMMAND`).
- `<trigger_moos_var mandatory_content="CommandType=GoTo">OUTGOING_COMMAND</trigger_mo` `OUTGOING_COMMAND` is the trigger variable and it must contain the substring `CommandType=GoTo`. That is, other commands might be published here (e.g. `CommandType=Loiter`, `CommandType=Track`) and we do not define the message structure of those here (this particular `<message></message>` is only for a GoTo message). other messages can be created to encode/decode these other command types.
- `<size>30</size>`: we want this message to fit in a WHOI micromodem FSK frame (32 bytes) and thus we have 30 bytes to work with (pAcommsHandler needs 2 bytes of header).

5. Finally, we fill out the `<publish></publish>` section which indicates where (i.e. what moos variables) and how (what format and which part(s) of the message) pGeneralCodec should publish decoded messages upon receipt of hex from other vehicles. Each `<publish></publish>` indicates a separate action that is taken upon receipt of a message. As many `<publish></publish>` sections as desired may be included for a given message. So, for our example message, we want to replicate the original string (a common practice):

```
INCOMING_COMMAND: CommandType=GoTo,goto_x=351,goto_y=294,
                  lights_on=true,special_instructions=make_toast,goto_speed=2.3
```

to do this we fill out a publish `<all />`. This is the simplest form of the `<publish></publish>` section:

```
<publish>
  <moos_var>INCOMING_COMMAND</moos_var>
  <all />
</publish>
```

this says to take every *message_var* and make a "key=value" comma-delimited string from it. the above `<publish></publish>` block is a shortcut for a much longer form:

```
<publish>
  <moos_var>INCOMING_COMMAND</moos_var>
  <format>type=goto,goto_x=%1%,goto_y=%2%,lights_on=%3%,
  special_instructions=%4%,goto_speed=%5%</format>
  <message_var>goto_x</message_var>
  <message_var>goto_y</message_var>
  <message_var>lights_on</message_var>
  <message_var>special_instructions</message_var>
  <message_var>goto_speed</message_var>
</publish>
```

these two blocks are functionally identical.

We may want to also publish the `special_instructions` to another moos variable, so that:

```
SPECIAL_INSTRUCTIONS: special_instructions=make_toast,lights_on=true
```

we can do this with another publish block:

```
<publish>
  <moos_var>SPECIAL_INSTRUCTIONS</moos_var>
  <format>special_instructions=%1%,lights_on=%2%</format>
  <message_var>new_instructions</message_var>
  <message_var>lights_on</message_var>
</publish>
```

in this case the `<format></format>` block is necessary because the default would be `<format>new_instructions=%1%,lights_on=%2%</format>` not `<format>special_instructions=%1%,lights_on=%2%</format>`.

Those are the basics to designing a **publish** triggering message.

Designing a time triggered message Scenario: we need a status message that grabs data from various moos variables and publishes them (encoded) on a time interval. We will not go into as much detail here, but rather highlight the changes from the previous scenario.

- you will notice

  ```
  <trigger>time</trigger>
  <trigger_time>30</trigger_time>
  ```

  instead of

  ```
  <trigger>publish</trigger>
  <trigger_moos_var mandatory_content="CommandType=GoTo">OUTGOING_COMMAND</trigger_moos_var>
  ```

  this indicates that a message should be made on a time interval (given by `<trigger_time></trigger_time>` which is every 30 seconds here), rather than on a publish to some moos variable.

- you will notice that all the *message_var*s have a `<moos_var></moos_var>` tag, which was omitted in the previous example since we were taking data from the trigger variable. Obviously, there is no trigger variable now so we must specify a location for the data to come from (in the moos db). The newest available value will be used when the message needs to be made. This means there is no guarantee that the data is fresh. Thus, you should use moos variables that are often updated for a `<trigger>time</trigger>` message. If this is not the case, a `<trigger>publish</trigger>` message (see previous scenario) may be a better choice.

- the format of the value read from the `<moos_var></moos_var>` can have several options. First, if the *message_var* is of a numeric type (`<int></int>`, `<float></float>`, `<bool></bool>`) and the `<moos_var></moos_var>` is a double, the value of the double is used as is. If the *message_var* is a string, two options are available. First, the pGeneralCodec looks for a substring of the form:

  ```
  name=value
  ```

  within the string and picks out the value to send for the message. If there is no such `name=` substring, the entire string is converted to the appropriate form. An example: we have a `<float></float>` called `<name>my_float</name>` that has a tag `<moos_var>SOME_FLOAT_VARIABLE</moos_var>`:

– if

`(double)SOME_FLOAT_VARIABLE: 3.56`

then 3.56 is sent.

– if instead

`(string)SOME_FLOAT_VARIABLE: "my_float=3.56"`

then 3.56 is still sent.

– if instead

`(string)SOME_FLOAT_VARIABLE: "3.56"`

again, 3.56 is sent.

– Finally, if some other string like

`(string)SOME_FLOAT_VARIABLE: "blah=3.56"`

then `blah=3.56` is converted (using streams) to a float, which will probably be zero or something else undesired. In other words, this case is not what you want, whereas the above three are fine.

3.8.6 Details

- the code is split into the following classes:

  1. `CMOOSApp` extended class `CpGeneralCodec` in `pGeneralCodec.cpp`. This is the moos app that you run.
  2. `Message` in `message.cpp`. This class defines everything you would want to do with a given message (`SENSOR_DEPLOY`, etc). then i instantiate as many Messages as needed (for every `<message></message>` tag in the XML file), read in the parameters with the Xerces XML parser, and call Message.encode() to create a message, or Message.decode() to decode it.
     (a) `MessageVar` in `message_var.cpp`. This class defines a *message_var* (i.e. a piece of the message (int, bool, etc.)) and any actions to be done to it. an instantiation is made for all the *message_vars* in each message. (e.g. `<int></int>`, `<string></string>`, etc.)
     (b) `Publish` in `publish.cpp`. This defines a given publish action upon receipt of a message. One instantiation for each `<publish></publish>` given.
  3. `MessageParser` in `message_xml_parser.cpp`. Reads in an XML file uses xerces.
  4. `MessageContentHandler` in `message_xml_callbacks.cpp`. Contains callbacks which populate the Message with data from the XML file.

- We may want to know the actual layout of the binary/hex message. Let us explain it with an example; for the first example message in `example1.xml` given in section 3.8.5, if we run pGeneralCodec we get information about that message:

```
type (static):
        value: {goto}
        size [bits]: [0]
goto_x (int):
        source: {OUTGOING_COMMAND}
        [min, max] = [0,10000]
        size [bits]: [14]
goto_y (int):
        source: {OUTGOING_COMMAND}
        [min, max] = [0,10000]
        size [bits]: [14]
lights_on (bool):
        source: {OUTGOING_COMMAND}
        size [bits]: [1]
new_instructions (string):
        source: {SPECIAL_INSTRUCTIONS}
        max_length: {10}
        size [bits]: [80]
goto_speed (float):
        source: {OUTGOING_COMMAND}
        [min, max] = [0,3]
        precision: {2}
        size [bits]: [9]
```

the calculated sizes are used to pack the message like so (`{#}` equals size of field in bits), where left to right is the same as reading the hex string from left to right:

```
[[0 {122}][goto_x {14}][goto_y {14}][lights_on {1}][new_instructions {80}][goto_speed {9}]]
```

where `[0 {122}]` means zero fill the front of the message to the full size (30 bytes = 240 bits minus 118 for other fields = 122). Byte boundaries are dissolved and encoded as a string "ABCDEF..." where the most significant byte (MSB, or leftmost 8 bits) is 0xAB, second MSB is 0xCD, etc. Encoding and decoding are done by functions available in `binary.h` in `libtes_util.a`.

3.8.7 Further examples

- I currently store our working message files in `moos-ivp-local/data/acomms`. look for .xml files in this directory for further examples.
- Probably the simplest message you can make (for a single string MOOS variable that gets truncated at 30 chars and sent to broadcast):

```
<?XML version="1.0" encoding="ASCII" standalone="yes"?>
<message_set>
  <message>
    <name>SimpleStringSender</name>
    <outgoing_hex_moos_var>OUT_STRING_HEX</outgoing_hex_moos_var>
    <incoming_hex_moos_var>IN_STRING_HEX</incoming_hex_moos_var>
    <trigger>publish</trigger>
```

```
      <trigger_moos_var>MY_STRING</trigger_moos_var>
      <size>30</size>
      <layout>
        <string>
          <name>my_string</name>
          <max_length>30</max_length>
        </string>
      <publish>
        <moos_var>INCOMING_COMMAND</moos_var>
        <all />
      </publish>
    </message>
  </message_set>
```

3.8.8 Message XML reference sheet

This is a quick reference of all the allowed tags with some comments about their use:

```
<?xml version="1.0" encoding="UTF-8"?> <!-- required XML declaration -->
<message_set> <!-- start the message_set. only one per file -->
  <message><!-- start the message. as many as desired per file -->
    <name></name> <!-- overall message name, only for human use -->

    <destination_moos_var key=""></destination_moos_var> <!-- if included, here is where you
                                                              find the destination of the message.
                                                              otherwise, use broadcast (0) -->
    <outgoing_hex_moos_var></outgoing_hex_moos_var> <!-- publish encoded hex string here -->
    <incoming_hex_moos_var></incoming_hex_moos_var> <!-- look for incoming hex string here -->

    <trigger>publish</trigger><!-- make a message (encode) when a write to a variable is noticed ...
    <trigger_moos_var mandatory_content=""></trigger_moos_var>
         <!-- ... specifically a write to THIS variable,
          which must contain ‘mandatory_content‘ within its string -->

    <!-- OR -->
    <trigger>time</trigger><!-- make a message on some regular time interval ... -->
    <trigger_time></trigger_time><!-- ...defined (in seconds) here -->

    <size>30</size>    <!-- enforce this message size (bytes) -->

    <!-- encoding -->
    <layout> <!-- start defining the message layout -->
      <static algorithm=""> <!-- a variable that is not actually sent, its value is given here
                                ’algorithm’ specifies one or more functions to be performed on
                                the data before encoding-->
        <name></name> <!-- name of this message variable (message_var) -->
        <value></value><!-- value of this static -->
      </static>
      <int algorithm=""><!-- stores signed or unsigned integers up to size long (typically 32) bits
        <moos_var key=""></moos_var> <!-- what moos variable to pull the value
                                          from when encoding. if omitted, use trigger_var
```

```
                                        'key' indicates the substring to look for in a key=value
                                        pair in the contents of this moos variable-->
    <name></name>
    <max></max><!-- maximum value this field can take. larger values are fixed to this limit -->
    <min></min><!-- minimum value this field can take. smaller values are fixed to this limit --
  </int>
  <float algorithm=""><!-- stores float variables up to size of the system long (typically 32 bi
    <moos_var key=""></moos_var> <!-- if omitted, use trigger_var -->
    <name></name>
    <max></max>
    <min></min>
    <precision></precision><!-- number of decimal places to keep -->
  </float>
  <bool algorithm=""> <!-- true or false -->
    <moos_var key=""></moos_var> <!-- if omitted, use trigger_var -->
    <name></name>
  </bool>
  <enum algorithm=""> <!-- a set of predefined strings -->
    <moos_var key=""></moos_var> <!-- if omitted, use trigger_var -->
    <name></name>
    <value></value> <!-- define the first enum string here -->
    <value></value> <!-- and the second here ... -->
    <value></value> <!-- and so on as long as you want ... -->
  </enum>
  <string algorithm="">
    <moos_var></moos_var> <!-- if omitted, use trigger_var -->
    <name></name>
    <max_length></max_length> <!-- truncate string to this number of chars -->
  </string>
</layout>

<!-- decoding -->
<publish> <!-- simple -->
  <moos_var type=""></moos_var> <!-- where to publish to. type="string" (default) or "double" --
  <format></format> <!-- format string. use boost ('%1%'', etc) or printf('%d', etc).
                    if omitted, use 'name=%1%,name=%2% etc for all message_var specified.
                    exception, a single message_var is specified, '%1%' is used for the format s
                    (that is, the key is not published) -->
  <message_var algorithm=""></message_var> <!-- name of the message_var to insert first.
                                                algorithm (optional) lets you perform
                                                conversions on the data before inputting to
                                                format string-->
  <message_var algorithm=""></message_var> <!-- name of the second message_var to insert
                                                ... and so on -->
</publish>

<publish> <!-- shortcut for all -->
  <moos_var></moos_var>
  <all /> <!-- equivalent to <message_var></message_var>
               entries for all message_var fields above -->
</publish>
```

```
  </message>
</message_set>
```

3.8.9 Glossary

- *message_var*: the term for a field within a message. a *message_var* can be of several types: int, bool, double, float, string, or enum. the *message_var*s are defined within the `<layout></layout>` part of the message.
- *modem id*: the number given to each whoi micromodem (this is like a variable MAC address) that defines senders and receivers. modem ids must be unique for each network and are configured using `$CCCFG,SRC,#`, where `#` is the modem id (integer from 0 to 127?).
- *publish*: the term for a given action to be performed upon receipt of a message. this term is used since this action will involve a moos publish to some variable (after some string parsing/formatting).
- *XML*: extensible markup language: a specification for defining a custom markup language, which is a set of annotations given to text to indicate structure. here we use XML to indicate the structure of a "message" and its subsequent breakup ("publishes", during decoding)
  - *tag*: the name given to the "metadata" in the XML file. for example: the tag `<message></message>` indicates the start and end of a message (this is "metadata"), but nothing about what it contains (which is"data")
  - *attributes*: data contained within a tag. for example, in `<int algorithm="to_upper"></int>`, **algorithm** is an attribute of the tag **int**.
  - *CDATA*: **C**haracter **DATA**, or the data contained within a tag or an attribute. for example, in `<name>nav_x</name>`, **nav_x** is CDATA.

3.9 pAcommsHandler

3.9.1 Brief Overview

problem acoustic communications (in our case, with the WHOI micromodem) are highly limited in throughput. thus, it is unreasonable to expect "total throughput" of all communications data. furthermore, even if total throughput is achievable over time, certain messages (e.g. vehicle status) have a lower tolerance for delay than others (e.g. CTD sample data). reference *http://acomms.whoi.edu/umodem/documentation.html* for more information on the WHOI micromodem.

pAcommsHandler roughly performs the same functions of pRouter but generalized to handle any number of message queues and extended to give more control over queue parameters. figure 2 gives a rough view of the current acoustic communications stack from the viewpoint of the operations carried out by the Laboratory for autonomous marine Sensing.

solution pAcommsHandler:

- maintains an arbitrary number of message queues (each tied to a different MOOS variable) for hexadecimal data strings (for the WHOI micromodem driver process, currently iMicroModem, to consume upon request)
- allows configuration of the queue priorities and dynamic growth of the priority over the time since the last sent message
- allows management of WHOI CCL message types as well as internal MOOS routing.

3.9.2 usage

compilation pAcommsHandler depends on the boost string library in addition to the libraries included in MOOS and moos-ivp-local. reference *http://www.boost.org/* or look for your distributions boost developer package (libboost-dev in debian/ubuntu).

3.9.3 Parameters for the pAcommsHandler Configuration Block

example moos file here is an example .moos file from the GLINT08 experiment in pianosa, italy. this particular file was used by the BF21 AUV 'Unicorn':

```
ProcessConfig = pAcommsHandler
{
  // available to all moos processes.
  AppTick    = 4
  CommsTick  = 4

  // available to all tes moos processes

  // verbose, terse, quiet
  verbosity = verbose
```

```
  // all case insensitive

  modem_id = 3

  // information about iMicroModem
  micromodem_command_var = MICROMODEM_COMMAND
  micromodem_data_var = MICROMODEM_DATA

  // send, send_CCL =
      // VarName
      //                              VariableID
      //                                  [Ack]
      //                                      [BlackoutTime]
      //                                          [MaxQueue]
      //                                               [NewestFirst]
      //                                                   [Priority]
      //                                                        [PriorityTimeConstant]

  send = OUT_CTD_HEX_30B,           1,  0,  0,  100, 1,  0.3, 120
  send = OUT_CTD_HEX_62B,           2,  0,  0,  100, 1,  0.3, 120
  send = OUT_CTD_HEX_254B,          3,  0,  0,  100, 1,  0.2, 120
  send = OUT_BTR_HEX_254B,          4,  0,  0,  100, 1,  0.5, 120

      //                            CCLIdentifierByte
  send_CCL = OUT_DEPLOY_HEX_32B,    1a, 1,  0,  1,   1,  10,  120
  send_CCL = OUT_PROSECUTE_HEX_32B, 1a, 1,  0,  1,   1,  10,  120

  send_CCL = OUT_STATUS_HEX_32B,    1b, 0,  30, 1,   1,  1,   120
  send_CCL = OUT_CONTACT_HEX_32B,   1b, 0,  0,  1,   1,  2,   120
  send_CCL = OUT_TRACK_HEX_32B,     1b, 0,  0,  1,   1,  4,   120

  // receive = VarName, VariableID
  receive = IN_CTD_HEX_30B, 1
  receive = IN_CTD_HEX_62B, 2
  receive = IN_CTD_HEX_254B, 3
  receive = IN_BTR_HEX_254B, 4

  // receive_CCL = VarName, CCLIdentifierByte
  receive_CCL = IN_PLUS_COMMANDS_HEX_32B, 1a
  receive_CCL = IN_PLUS_MESSAGES_HEX_32B, 1b
}
```

filling out the .moos file

general parameters

- `verbosity`: choose verbose for full text terminal output, terse for symbolic heartbeat output, and quiet for no terminal output.
- `modem_id`: integer that specifies the modem id of this current vehicle / community. this must match the micromodem SRC configuration parameter (send the modem $CCCFQ,SRC to check). for the remainder of the document, "modem ID" refers to the value $CCCFG,SRC,modem_id

iMicroModem parameters

- `micromodem_command_var`: the variable that iMicroModem is configured to receive commands on. this is the value in the iMicroModem .moos block `VarNamePrefix` concatenated to "_COMMAND". for example if `VarNamePrefix=MM`, then set `micromodem_command_var=MM_COMMAND`
- `micromodem_data_var`: the variable that iMicroModem is configured to send data on. this is the value in the iMicroModem .moos block `VarNamePrefix` concatenated to "_DATA". for example if `VarNamePrefix=MM`, then set `micromodem_command_var=MM_DATA`

outgoing queue parameters

- `send`: configure a queue for messages to be sent from this community. the send parameter is a comma delimited string of two mandatory parameters and up to six optional parameters (in order specified here). to skip an optional field and use the default value, simply use a blank ",":
  - `VarName`: name of the moos variable to subscribe to for messages to add to this queue. publishes here should be pure hexadecimal or a key=value string specified later in section 3.9.4.
  - `VariableID`: a user specified integer from 0-15 that is a tag for the VarName. thus, each `VarName` must have a unique `VariableID`. only the `VariableID` is sent with the message (to save space), not the full `VarName` of the queue. thus, this must match the `VariableID` on the receiving vehicle's `receive` configuration in the incoming parameters section below. for example, if i have `send=SOME_OUT_HEX, 1` on the sending vehicle, the receiving vehicles must have a field `receive=SOME_IN_HEX, 1`. all messages with ID 1 will be put in `SOME_IN_HEX`. clearly, if unique mapping is desired on the receiving end, unique `VariableID`s must be used on the sending end.
  - `Ack`: boolean flag (1=true, 0=false) whether to request an acoustic acknowledgment on all sent messages from this field. if omitted, default of 0 (false, no ack) is used.
  - `BlackoutTime`: time in seconds after sending a message from this queue for which no more messages will be sent. use this field to stop an always full queue from hogging the channel. if omitted, default of 0 (no blackout) is used.
  - `MaxQueue`: number of messages allowed in the queue before discarding messages. if `NewestFirst` is set to true, the oldest message in the queue is discarded to make room for the new message. otherwise, any new messages are disregarded until the space in the queue opens up.
  - `NewestFirst`: boolean flag (1=true=FILO, 0=false=FIFO) whether to send newest messages in the queue first (FILO) or not (FIFO).

– `Priority`: base priority value for this message queue. priorities are calculated on a request for data by the modem (to send a message). the queue with the highest priority (and isn't in blackout) is chosen. the actual priority (P) is calculated by

$$P = P_{base} \exp\left[(t - t_{last})/\tau\right]$$

where P_{base} is the value set here, t is the current time (in seconds), t_{last} is the time of the last send from this queue, and τ is the `PriorityTimeConstant`. essentially, a message with low `PriorityTimeConstant` will become effective quickly again after a sent message (the exponential grows faster). a higher base priority will also increase the effectiveness of the queue.

- `send_CCL`: configure a queue for a WHOI CCL message type (do not use the moos `VariableID`). the queues mentioned above (`send=`) use the CCL identifier byte 0x20 and then the second byte includes the `VariableID` leaving $N - 2$ bytes for the user where $N = 32, 64,$ or 256 depending on the modem rate. this queue `send_CCL` does not use the second byte, thus making it useful for handling the static CCL types defined by WHOI. all parameters in the `send_CCL` list are the same as those for the `send` list with the exception of the second parameter. here it is the `CCLIdentifierByte` in hexadecimal.

incoming parameters

- `receive`: specifies a mapping between an incoming `VariableID` and a moos variable to place the incoming message. the sender's modem id is stored as the community name for the received message.
- `receive_CCL`: same as `receive` except for WHOI CCL types. rather than a `VariableID` you specify a `CCLIdentifierByte` that is the first byte of the CCL message.

3.9.4 MOOS variables subscribed to by pAcommsHandler:

All variables are configurable in the .moos file as described in section 3.9.3. For example, if `send = OUT_CTD_HEX_30B, ...` is set in the .moos file, then `OUT_CTD_HEX_30B` is the variable actually subscribed.

.moos file key specifying a MOOS variable	Type	Description	Published by
micromodem_data_var	$	micromodem data line (data requests also come on this variable)	iMicroModem
send=varname,...	$	varname contains hexadecimal string to queue (and eventually send).	Various Codecs (pGeneralCodec, pCTDCodec, etc.)

input (to pAcommsHandler) formats pAcommsHandler accepts several formats for the strings placed in the various `VarName` moos variables (outgoing message queues). the simplest is just a hexadecimal string of the appropriate length ($N - 2$ bytes where $N = 32, 64,$ or 256 depending on the modem rate). this message will be queued to send to broadcast (modem id 0[2]). an example:

[2]the WHOI micromodem does not treat certain modem IDs specially (such as zero). all data are reported to the control computer, regardless of whether that machine is the intended recipient. however, the communications structure

```
OUT_CTD_HEX_30B: 6850a232ccd2be62e1f36e6b02fffa92385bce8fa2109efa8902ddf3a02
```

would queue a CTD message to be sent to broadcast (modem ID 0), given the above configuration file.

pAcommsHandler also accepts a string of key=value, comma delimited fields allowing for more flexibility. currently, the only fields supported are `HexData=` (required, of course) and `Dest=` for the intended destination modem_id.

```
OUT_CTD_HEX_30B: Dest=3,HexData=6850a232ccd2be62e1f36e6b02fffa92385bce8fa2109efa8902ddf3a02
```

would queue a CTD message to be sent to id 3.

3.9.5 MOOS variables subscribed to by pAcommsHandler:

Similarly to the subscriptions, the publishes done by pAcommsHandler are all defined in the .moos file. Again, instead of MOOS variables, the table below indicates the .moos file keys for which one can define the publishes.

.moos file key specifying a MOOS variable	Type	Description	Format
micromodem_command_var	$	commands to iMicroModem	see below
receive=varname,...	$	varname contains a hexadecimal string from the micromodem to route to this queue.	see below

output (from pAcommsHandler) formats Upon receipt of a message from the micromodem driver, the first byte of the message is checked against the moos CCL type (0x20) and any of the additional `receive_CCL CCLIdentifierByte` given in the .moos file. if none of these match, the message is disregarded. if the message matches the moos CCL type (0x20), the second byte is examined for the `VariableID`. if the `VariableID` matches one of the `receive` fields in the .moos file, the hexadecimal string is stripped of its first two bytes and placed in the correspoding `VarName` moos variable. the community name is set to the sender's variable ID [3]. for example, the modem passes to iMicroModem:[4]

```
$CARXD,3,0,0,1,20016850a232ccd2be62e1f36e6b02fffa92385bce8fa2109efa8902ddf3a02
```

iMicroModem [5] passes this message to pAcommsHandler, which strips the two 0x2001 bytes, and places the message in the appropriate moos variable, which for the example .moos file here is `IN_CTD_HEX_30B`:

```
IN_CTD_HEX_30B: 6850a232ccd2be62e1f36e6b02fffa92385bce8fa2109efa8902ddf3a02
{Community=3}
```

---

defined by pAcommsHandler and other *tes* processes treat modem ID 0 as broadcast (much like xxx.xxx.xxx.255 is used for broadcast on TCP/IP networks). this means that incoming messages are read if they are to our modem ID or to modem ID 0.

[3] use CMOOSMsg::GetCommunity() to extract the community

[4] see Micro-Modem Mainboard Software Interface Guide at http://acomms.whoi.edu/umodem/documentation.html for details on these messages

[5] it is my hope to replace this process shortly so minimal details are given on iMicroModem in this documentation

3.9.6 details

here we go. rather than a mess more prose, i will resort to another bulleted list of things you may want to know. this program is a bit fluid at the moment so expect additions and changes that this document may or may not keep pace with. when in doubt an email is (almost!) always promptly responded to.

- more details on what we're doing here: pAcommsHandler takes all the configured queues and maintains a stack of messages for each queue. when it is prompted by data by iMicroModem, it has a priority "contest" between the queues. the queue with the current highest priority (as determined by the `Priority` and `PriorityTimeConstant` fields) is selected. the next message in that queue is then provided to iMicroModem to send. for modem messages with multiple frames per packet, each frame is a separate contest. thus a single packet may contain frames from different queues (e.g. a rate 5 PSK packet has eight 256 byte frames. frame 1 might grab a STATUS message since that has the current highest queue. then frame 2 may grab a BTR message and frames 3-8 are filled up with CTD messages (e.g. STATUS is in blackout, BTR queue is empty)).

- i mentioned this above but it's worth going over again. we choose modem id 0 as broadcast. thus messages with the destination field = 0 will always be read by all nodes and reported to the appropriate moos variable. otherwise, we ignore messages unless they correspond to our modem id. so if you send a message to modem id 10, pAcommsHandler for modem ids $1 \rightarrow 9$, $11 \rightarrow N$ will ignore that. this is not the default behavior of the modem, which always reports data, regardless of the sender's ID.

- if you do not wish for dynamic growth of the priorities, simply set the `PriorityTimeConstant` to a very large value such as $1e300$. then the priorities grow as

$$P = P_{base} \exp\left[(t - t_{last})/1e300\right] \simeq P_{base} \exp\left(0\right) = P_{base}$$

- for messages with `Ack=1` (acknowledge requested), the last message continues to be re-sent (that is, it is not popped from the message queue) until the ACK is received from the modem (thus blocking the sending of other messages). perhaps i will add max retries at some point soon. messages with `Ack=0` are popped and discarded when they are sent (no retries).

- this isn't as pretty as could be, but if you need to know the destination of the next message to be sent (i.e. the next message in the queue with the present highest priority), you can send a message to moos variable `POLLER_DEST_REQUEST` (contents of the message doesn't matter. you will get a reply (double) in `POLLER_DEST_COMMAND` (e.g. the next message should go to ID 5):

  ```
  POLLER_DEST_COMMAND: 5
  ```

  as the names imply, this is currently used by pAcommsPoller.

- if all else fails, read the terminal output. it's not so bad.

3.10 pAcommsPoller

3.10.1 Brief Overview

This MOOS module does a darn good job at doing what it is supposed to do

3.10.2 Parameters for the pAcommsPoller Configuration Block

3.10.3 MOOS variables subscribed to by pAcommspoller:

.

3.10.4 MOOS variables published by pAcommsPoller:

.

3.11 iMicroModem

3.11.1 Brief Overview

This MOOS module does a darn good job at doing what it is supposed to do

3.11.2 Parameters for the pAcommsPoller Configuration Block

3.11.3 MOOS variables subscribed to by iMicroModem:

.

3.11.4 MOOS variables published by iMicroModem:

.

A Appendix - pGeneralCodec Configuration Files

The message set developed and applied in the ONR PLUSnet program used a dedicated set of CCL messages for transmitting *Deploy* and *Prosecute Commnds*, and *Status, Contact* and *Track Reports*. The CCL message decoding was performed using the MOOS process `pFramer`. In the fall of 2008, Toby Schneder at MIT developed a new, totally generic message coder-decoder, `pGeneralCodec`, replacing the highly restrictive `pFramer`, using a special Dynamic CCL (D-CCL) message with the official CCL type 32. The message coding/decoding scheme is specified at run-time in a configuration file using the xml macro format, thus allowing for a much more flexible message set, easily modified and expanded without re-coding. For backward compatibility, the PLUSNet message format has been retained using the new Codec, specfied in two `xml` configuration files, referenced in the `MOOS` configuration block for `pGeneralCodec`, shown in Listing A.1.

Listing A.1 - MIT-LAMS prototype `pGeneralCodec` *configuration block .*

```
 1 ProcessConfig = pGeneralCodec
 2 {
 3 AppTick    = 4
 4 CommsTick  = 4
 5
 6   verbosity = verbose
 7
 8 // located in moos-ivp-local/data/acomms/
 9   message_file = ../../../data/acomms/nafcon_command.xml
10   message_file = ../../../data/acomms/nafcon_report.xml
11 // henriks attempt at creating message for TARGET_STATE for simulator
12   message_file = ../../../data/acomms/nafcon_targetsim.xml
13 }
```

The message files contain the coding/decoding scheme in xml, as described in Section 3.8. For convenience the configurations for commands and reports are specified in separate files, `nafcon_commands.xml` and `nafcon_reports.xml`. The structure of the two files is shown in Listings A.2 and A.3.

Listing A.2 pGeneralCodex Configuration file `nafcon_commands.xml` *for Sensor Commands*

```
<?xml version="1.0" encoding="ASCII" standalone="yes"?>
<!-- contains nafcon (plus) commands: deploy, prosecute -->
<message_set>
   ...
   DEPLOY Command XML block
   ...
   PROSECUTE Command XML block
   ...
</message_set>
```

Listing A.3 pGeneralCodex Configuration file `nafcon_reports.xml` *for Sensor Reports*

```
<?xml version="1.0" encoding="ASCII" standalone="yes"?>
<!-- contains nafcon (plus) reports: status, contact and track -->
<message_set>
   ...
   STATUS Report XML block
   ...
   CONTACT Report XML block
   ...
   TRACK Report XML block
   ...
</message_set>
```

The last message file in the configuration block, `nafcon_targetsim` is a message specifically designed for simulating acoustic sources synchronously on all nodes. This message is used in conjunction with the target GUI `PassiveTgtSim` in the topside MOOS community for transmitting the target parameters to all nodes via the real or virtual modem network. The current XML configuration file `nafcon_targetsim` is shown in Listing A.9.

Listing A.4 pGeneralCodex XML Configuration Block for Sensor Deploy Command

```
...
<message>
  <name>SENSOR_DEPLOY</name>
  <destination_moos_var key="DestinationPlatformId">
    PLUSNET_MESSAGES
  </destination_moos_var>
  <outgoing_hex_moos_var>
    OUT_DEPLOY_HEX_30B
  </outgoing_hex_moos_var>
  <incoming_hex_moos_var>
    IN_DEPLOY_HEX_30B
  </incoming_hex_moos_var>
  <trigger>publish</trigger> <!-- publish to moos var -->
  <trigger_moos_var mandatory_content=
    "MessageType=SENSOR_DEPLOY">
    PLUSNET_MESSAGES</trigger_moos_var>
  <size>30</size>
  <layout>
    <static>
      <name>MessageType</name>
      <value>SENSOR_DEPLOY</value>
    </static>
    <static>
      <name>SensorCommandType</name>
      <value>0</value>
    </static>
    <int>
      <name>SourcePlatformId</name>
      <max>31</max>
      <min>0</min>
    </int>
    <int>
      <name>DestinationPlatformId</name>
      <max>31</max>
      <min>0</min>
    </int>
    <int>
      <name>Timestamp</name>
      <max>2000000000</max>
      <min>1000000000</min>
    </int>
    <float>
      <name>DeployLatitude</name>
      <precision>5</precision>
      <max>90</max>
      <min>-90</min>
    </float>
    <float>
      <name>DeployLongitude</name>
      <precision>5</precision>
      <max>180</max>
      <min>-180</min>
    </float>
    <int>
      <name>DeployDepth</name>
      <max>255</max>
      <min>0</min>
    </int>
    <int>
      <name>DeployDuration</name>
      <max>63</max>
      <min>1</min>
    </int>
    <float>
      <name>AbortLatitude</name>
      <precision>4</precision>
      <max>90</max>
      <min>-90</min>
    </float>
    <float>
      <name>AbortLongitude</name>
      <precision>4</precision>
      <max>180</max>
      <min>-180</min>
    </float>
    <int>
      <name>AbortDepth</name>
      <max>511</max>
      <min>0</min>
    </int>
    <int>
      <name>MissionType</name>
      <max>7</max>
      <min>0</min>
    </int>
    <int>
      <name>OperatingRadius</name>
      <max>1023</max>
      <min>0</min>
    </int>
    <float algorithm="angle_0_360">
      <name>DCLFOVStartHeading</name>
      <max>360</max>
      <min>0</min>
      <precision>1</precision>
    </float>
    <float algorithm="angle_0_360">
      <name>DCLFOVEndHeading</name>
      <max>360</max>
      <min>0</min>
      <precision>1</precision>
    </float>
    <float>
      <name>DCLSearchRate</name>
      <max>255</max>
      <min>0</min>
      <precision>1</precision>
    </float>
  </layout>
  <!-- decoding -->
  <publish>
    <moos_var>NAFCON_MESSAGES</moos_var>
    <all />
  </publish>
</message>
...
```

Listing A.5 pGeneralCodec XML Configuration Block for Prosecute Command

```
...
<message>
  <name>SENSOR_PROSECUTE</name>
  <outgoing_hex_moos_var>
    OUT_PROSECUTE_HEX_30B
  </outgoing_hex_moos_var>
  <incoming_hex_moos_var>
    IN_PROSECUTE_HEX_30B
  </incoming_hex_moos_var>
  <destination_moos_var key="DestinationPlatformId">
    PLUSNET_MESSAGES
  </destination_moos_var>
  <trigger>publish</trigger> <!-- publish to moos var -->
  <trigger_moos_var
    mandatory_content="MessageType=SENSOR_PROSECUTE">
    PLUSNET_MESSAGES
  </trigger_moos_var>
  <size>30</size>
  <layout>
    <static>
      <name>MessageType</name>
      <value>SENSOR_PROSECUTE</value>
    </static>
    <int>
      <name>SensorCommandType</name>
      <min>1</min>
      <max>4</max>
    </int>
    <int>
      <name>SourcePlatformId</name>
      <max>31</max>
      <min>0</min>
    </int>
    <int>
      <name>DestinationPlatformId</name>
      <max>31</max>
      <min>0</min>
    </int>
    <int>
      <name>TargetTimestamp</name>
      <max>2000000000</max>
      <min>1000000000</min>
    </int>
    <int>
      <name>PlatformID</name>
      <max>31</max>
      <min>0</min>
    </int>
    <int>
      <name>TrackNumber</name>
      <max>255</max>
      <min>0</min>
    </int>
    <float>
      <name>TargetLatitude</name>
      <precision>5</precision>
      <max>90</max>
      <min>-90</min>
    </float>
    <float>
      <name>TargetLongitude</name>
      <precision>5</precision>
      <max>180</max>
      <min>-180</min>
    </float>
    <int>
      <name>TargetDepth</name>
      <max>1023</max>
      <min>0</min>
    </int>
    <float algorithm="angle_0_360">
      <name>TargetHeading</name>
      <max>360</max>
      <min>0</min>
      <precision>1</precision>
    </float>
    <float>
      <name>TargetSpeed</name>
      <max>12.8</max>
      <min>0</min>
      <precision>1</precision>
    </float>
    <int>
      <name>TargetSpectralLevel1</name>
      <max>127</max>
      <min>0</min>
    </int>
    <int>
      <name>TargetFrequency1</name>
      <max>4095</max>
      <min>0</min>
    </int>
    <int>
      <name>TargetBandwidth1</name>
      <max>20</max>
      <min>0</min>
    </int>
    <int>
      <name>TargetSpectralLevel2</name>
      <max>127</max>
      <min>0</min>
    </int>
    <int>
      <name>TargetFrequency2</name>
      <max>4095</max>
      <min>0</min>
    </int>
    <int>
      <name>TargetBandwidth2</name>
      <max>20</max>
      <min>0</min>
    </int>
    <int>
      <name>ProsecuteDuration</name>
      <max>63</max>
      <min>1</min>
    </int>
    <float>
      <name>AbortLatitude</name>
      <precision>4</precision>
      <max>90</max>
      <min>-90</min>
    </float>
    <float>
```

```
      <name>AbortLongitude</name>
      <precision>4</precision>
      <max>180</max>
      <min>-180</min>
    </float>
    <int>
      <name>AbortDepth</name>
      <max>127</max>
      <min>0</min>
    </int>
  </layout>
  <!-- decoding -->
  <publish>
    <moos_var>NAFCON_MESSAGES</moos_var>
    <all />
  </publish>
</message>
...
```

Listing A.6 pGeneralCodex XML Configuration Block for Status Reports

```
...
<message>
  <name>SENSOR_STATUS</name>
  <outgoing_hex_moos_var>
    OUT_STATUS_HEX_30B
  </outgoing_hex_moos_var>
  <incoming_hex_moos_var>
    IN_STATUS_HEX_30B
  </incoming_hex_moos_var>
  <trigger>publish</trigger> <!-- publish to moos var -->
  <trigger_moos_var
        mandatory_content="MessageType=SENSOR_STATUS">
        PLUSNET_MESSAGES</trigger_moos_var>
  <size>30</size>
  <layout>
    <static>
      <name>MessageType</name>
      <value>SENSOR_STATUS</value>
    </static>
    <static>
      <name>SensorReportType</name>
      <value>0</value>
    </static>
    <int>
      <name>SourcePlatformId</name>
      <max>31</max>
      <min>0</min>
    </int>
    <int>
      <name>DestinationPlatformId</name>
      <max>31</max>
      <min>0</min>
    </int>
    <int>
      <name>Timestamp</name>
      <max>2000000000</max>
      <min>1000000000</min>
    </int>
    <float>
      <name>NodeLatitude</name>
      <precision>5</precision>
      <max>90</max>
      <min>-90</min>
    </float>
    <float>
      <name>NodeLongitude</name>
      <precision>5</precision>
      <max>180</max>
      <min>-180</min>
    </float>
    <int>
      <name>NodeDepth</name>
      <max>1023</max>
      <min>0</min>
    </int>
    <int>
      <name>NodeCEP</name>
      <max>1023</max>
      <min>0</min>
    </int>
    <float algorithm="angle_0_360">
      <name>NodeHeading</name>
      <max>360</max>
      <min>0</min>
      <precision>1</precision>
    </float>
    <float>
      <name>NodeSpeed</name>
      <max>12.8</max>
      <min>0</min>
      <precision>1</precision>
    </float>
    <int>
      <name>MissionState</name>
      <max>7</max>
      <min>0</min>
    </int>
    <int>
      <name>MissionType</name>
      <max>7</max>
      <min>0</min>
    </int>
    <int>
      <name>LastGPSTimestamp</name>
      <max>2000000000</max>
      <min>1000000000</min>
    </int>
    <int>
      <name>PowerLife</name>
      <max>1023</max>
      <min>1</min>
    </int>
    <int>
      <name>SensorHealth</name>
      <max>15</max>
      <min>0</min>
    </int>
    <int>
      <name>RecorderState</name>
      <max>1</max>
      <min>0</min>
    </int>
    <int>
      <name>RecorderLife</name>
      <max>31</max>
      <min>0</min>
    </int>
    <int>
      <name>NodeSpecificInfo0</name>
      <max>255</max>
      <min>0</min>
    </int>
    <int>
      <name>NodeSpecificInfo1</name>
      <max>255</max>
      <min>0</min>
    </int>
    <int>
      <name>NodeSpecificInfo2</name>
      <max>255</max>
      <min>0</min>
    </int>
    <int>
      <name>NodeSpecificInfo3</name>
```

```
    <max>255</max>
    <min>0</min>
  </int>
  <int>
    <name>NodeSpecificInfo4</name>
    <max>255</max>
    <min>0</min>
  </int>
  <int>
    <name>NodeSpecificInfo5</name>
    <max>255</max>
    <min>0</min>
  </int>
</layout>
<!-- decoding -->
<publish>
  <moos_var>NAFCON_MESSAGES</moos_var>
  <all />
</publish>
<!-- start pTransponderAIS replacement -->
<publish>
  <moos_var>AIS_REPORT</moos_var>
  <format>
    NAME=%1%,TYPE=%2%,UTC_TIME=%3$.0lf,
    X=%4%,Y=%5%,LAT=%6$lf,LON=%7$lf,
    SPD=%8%,HDG=%9%,DEPTH=%10%
  </format>
  <message_var algorithm="modem_id2name,to_lower">
    SourcePlatformId</message_var>
  <message_var algorithm="modem_id2type,to_lower">
    SourcePlatformId</message_var>
  <message_var>Timestamp</message_var>
  <message_var algorithm="lon2utm_x:NodeLatitude">
    NodeLongitude</message_var>
  <message_var algorithm="lat2utm_y:NodeLongitude">
    NodeLatitude</message_var>
  <message_var>NodeLatitude</message_var>
  <message_var>NodeLongitude</message_var>
  <message_var>NodeSpeed</message_var>
  <message_var>NodeHeading</message_var>
  <message_var>NodeDepth</message_var>
</publish>
<publish>
  <moos_var type="double">
    %1%_NAV_UTC
  </moos_var>
  <format>%2$lf</format>
  <message_var algorithm="modem_id2name,to_upper">
     SourcePlatformId</message_var>
  <message_var>Timestamp</message_var>
</publish>
<publish>
  <moos_var type="double">
    %1%_NAV_X
  </moos_var>
  <format>%2%</format>
  <message_var algorithm="modem_id2name,to_upper">
     SourcePlatformId</message_var>
  <message_var algorithm="lon2utm_x:NodeLatitude">
     NodeLongitude</message_var>
</publish>
<publish>
  <moos_var type="double">
    %1%_NAV_Y
  </moos_var>
  <format>%2%</format>
  <message_var algorithm="modem_id2name,to_upper">
     SourcePlatformId</message_var>
  <message_var algorithm="lat2utm_y:NodeLongitude">
     NodeLatitude</message_var>
</publish>
<publish>
  <moos_var type="double">
    %1%_NAV_LAT
  </moos_var>
  <format>%2%</format>
  <message_var algorithm="modem_id2name,to_upper">
    SourcePlatformId</message_var>
  <message_var>NodeLatitude</message_var>
</publish>
<publish>
  <moos_var type="double">
    %1%_NAV_LONG
  </moos_var>
  <format>%2$lf</format>
  <message_var algorithm="modem_id2name,to_upper">
     SourcePlatformId</message_var>
  <message_var>NodeLongitude</message_var>
</publish>
<publish>
  <moos_var type="double">
    %1%_NAV_SPEED
  </moos_var>
  <format>%2$lf</format>
  <message_var algorithm="modem_id2name,to_upper">
     SourcePlatformId</message_var>
  <message_var>NodeSpeed</message_var>
</publish>
<publish>
  <moos_var type="double">
    %1%_NAV_HEADING
  </moos_var>
  <format>%2%</format>
  <message_var algorithm="modem_id2name,to_upper">
     SourcePlatformId</message_var>
  <message_var>NodeHeading</message_var>
</publish>
<publish>
  <moos_var type="double">
    %1%_NAV_DEPTH
  </moos_var>
  <format>%2%</format>
  <message_var algorithm="modem_id2name,to_upper">
    SourcePlatformId</message_var>
  <message_var>NodeDepth</message_var>
</publish>
<!-- end pTransponderAIS replacement -->
</message>
...
```

Listing A.7 pGeneralCodex XML Configuration Block for Contact Report

```
...
<message>
  <name>SENSOR_CONTACT</name>
  <outgoing_hex_moos_var>
    OUT_CONTACT_HEX_30B
  </outgoing_hex_moos_var>
  <incoming_hex_moos_var>
    IN_CONTACT_HEX_30B
  </incoming_hex_moos_var>
  <trigger>publish</trigger>
  <trigger_moos_var
    mandatory_content="MessageType=SENSOR_CONTACT">
    PLUSNET_MESSAGES
  </trigger_moos_var>
  <size>30</size>
  <layout>
    <static>
      <name>MessageType</name>
      <value>SENSOR_CONTACT</value>
    </static>
    <static>
      <name>SensorReportType</name>
      <value>1</value>
    </static>
    <int>
      <name>SourcePlatformId</name>
      <max>31</max>
      <min>0</min>
    </int>
    <int>
      <name>DestinationPlatformId</name>
      <max>31</max>
      <min>0</min>
    </int>
    <int>
      <name>ContactTimestamp</name>
      <max>2000000000</max>
      <min>1000000000</min>
    </int>
    <float algorithm="angle_0_360">
      <name>SensorHeading</name>
      <max>360</max>
      <min>0</min>
      <precision>1</precision>
    </float>
    <float>
      <name>SensorPitch</name>
      <max>90</max>
      <min>-90</min>
      <precision>0</precision>
    </float>
    <float>
      <name>SensorRoll</name>
      <max>180</max>
      <min>-180</min>
      <precision>0</precision>
    </float>
    <float>
      <name>SensorLatitude</name>
      <precision>5</precision>
      <max>90</max>
      <min>-90</min>
    </float>
    <float>
      <name>SensorLongitude</name>
      <precision>5</precision>
      <max>180</max>
      <min>-180</min>
    </float>
    <int>
      <name>SensorDepth</name>
      <max>1023</max>
      <min>0</min>
    </int>
    <int>
      <name>SensorCEP</name>
      <max>1023</max>
      <min>0</min>
    </int>
    <float>
      <name>ContactBearing</name>
      <max>360</max>
      <min>0</min>
      <precision>1</precision>
    </float>
    <float>
      <name>ContactBearingUncertainty</name>
      <max>10</max>
      <min>0</min>
      <precision>1</precision>
    </float>
    <float>
      <name>ContactBearingRate</name>
      <max>10</max>
      <min>-10</min>
      <precision>1</precision>
    </float>
    <float>
      <name>ContactBearingRateUncertainty</name>
      <max>3.1</max>
      <min>0</min>
      <precision>1</precision>
    </float>
    <float>
      <name>ContactElevation</name>
      <max>90</max>
      <min>-90</min>
      <precision>1</precision>
    </float>
    <float>
      <name>ContactElevationUncertainty</name>
      <max>10</max>
      <min>0</min>
      <precision>1</precision>
    </float>
    <int>
      <name>ContactSpectralLevel1</name>
      <max>127</max>
      <min>0</min>
    </int>
    <int>
      <name>ContactFrequency1</name>
      <max>4095</max>
      <min>0</min>
```

```
  </int>
  <int>
    <name>ContactBandwidth1</name>
    <max>20</max>
    <min>0</min>
  </int>
  <int>
    <name>ContactSpectralLevel2</name>
    <max>127</max>
    <min>0</min>
  </int>
  <int>
    <name>ContactFrequency2</name>
    <max>4095</max>
    <min>0</min>
  </int>
  <int>
    <name>ContactBandwidth2</name>
    <max>20</max>
    <min>0</min>
  </int>
</layout>
<!-- decoding -->
<publish>
  <moos_var>NAFCON_MESSAGES</moos_var>
  <all />
</publish>
<!-- start pTransponderAIS replacement -->
<publish>
  <moos_var>AIS_REPORT</moos_var>
  <format>
    NAME=%1%,TYPE=%2%,UTC_TIME=%3$.0lf,
    X=%4%,Y=%5%,LAT=%6$lf,LON=%7$lf,
    SPD=,HDG=%8%,DEPTH=%9%
  </format>
  <message_var algorithm="modem_id2name,to_lower">
    SourcePlatformId</message_var>
  <message_var algorithm="modem_id2type,to_lower">
    SourcePlatformId</message_var>
  <message_var>ContactTimestamp</message_var>
  <message_var algorithm="lon2utm_x:SensorLatitude">
    SensorLongitude</message_var>
  <message_var algorithm="lat2utm_y:SensorLongitude">
    SensorLatitude</message_var>
  <message_var>SensorLatitude</message_var>
  <message_var>SensorLongitude</message_var>
  <message_var>SensorHeading</message_var>
  <message_var>SensorDepth</message_var>
</publish>
<publish>
  <moos_var type="double">
    %1%_NAV_UTC
  </moos_var>
  <format>%2$lf</format>
  <message_var algorithm="modem_id2name,to_upper">
    SourcePlatformId</message_var>
  <message_var>ContactTimestamp</message_var>
</publish>
<publish>
  <moos_var type="double">
    %1%_NAV_X
  </moos_var>
  <format>%2%</format>
  <message_var algorithm="modem_id2name,to_upper">
    SourcePlatformId</message_var>
  <message_var algorithm="lon2utm_x:SensorLatitude">
    SensorLongitude</message_var>
</publish>
<publish>
  <moos_var type="double">
    %1%_NAV_Y
  </moos_var>
  <format>%2%</format>
  <message_var algorithm="modem_id2name,to_upper">
    SourcePlatformId</message_var>
  <message_var algorithm="lat2utm_y:SensorLongitude">
    SensorLatitude</message_var>
</publish>
<publish>
  <moos_var type="double">
    %1%_NAV_LAT
  </moos_var>
  <format>%2%</format>
  <message_var algorithm="modem_id2name,to_upper">
    SourcePlatformId</message_var>
  <message_var>SensorLatitude</message_var>
</publish>
<publish>
  <moos_var type="double">
    %1%_NAV_LONG
  </moos_var>
  <format>%2$lf</format>
  <message_var algorithm="modem_id2name,to_upper">
    SourcePlatformId</message_var>
  <message_var>SensorLongitude</message_var>
</publish>
<publish>
  <moos_var type="double">
    %1%_NAV_HEADING
  </moos_var>
  <format>%2%</format>
  <message_var algorithm="modem_id2name,to_upper">
    SourcePlatformId</message_var>
  <message_var>SensorHeading</message_var>
</publish>
<publish>
  <moos_var type="double">
    %1%_NAV_DEPTH
  </moos_var>
  <format>%2%</format>
  <message_var algorithm="modem_id2name,to_upper">
    SourcePlatformId</message_var>
  <message_var>SensorDepth</message_var>
</publish>
<!-- end pTransponderAIS replacement -->
</message>
...
```

Listing A.8 pGeneralCodex XML Configuration Block for Track Report

```
...
<message>
  <name>SENSOR_TRACK</name>
  <outgoing_hex_moos_var>
    OUT_TRACK_HEX_30B
  </outgoing_hex_moos_var>
  <incoming_hex_moos_var>
    IN_TRACK_HEX_30B
  </incoming_hex_moos_var>
  <trigger>publish</trigger>
  <trigger_moos_var
    mandatory_content="MessageType=SENSOR_TRACK">
    PLUSNET_MESSAGES
  </trigger_moos_var>
  <size>30</size>
  <layout>
    <static>
      <name>MessageType</name>
      <value>SENSOR_TRACK</value>
    </static>
    <static>
      <name>SensorReportType</name>
      <value>2</value>
    </static>
    <int>
      <name>SourcePlatformId</name>
      <max>31</max>
      <min>0</min>
    </int>
    <int>
      <name>DestinationPlatformId</name>
      <max>31</max>
      <min>0</min>
    </int>
    <int>
      <name>TrackTimestamp</name>
      <max>2000000000</max>
      <min>1000000000</min>
    </int>
    <int>
      <name>PlatformID</name>
      <max>31</max>
      <min>0</min>
    </int>
    <int>
      <name>TrackNumber</name>
      <max>255</max>
      <min>0</min>
    </int>
    <float>
      <name>TrackLatitude</name>
      <precision>5</precision>
      <max>90</max>
      <min>-90</min>
    </float>
    <float>
      <name>TrackLongitude</name>
      <precision>5</precision>
      <max>180</max>
      <min>-180</min>
    </float>
    <int>
      <name>TrackCEP</name>
      <max>1023</max>
      <min>0</min>
    </int>
    <int>
      <name>TrackDepth</name>
      <max>1023</max>
      <min>0</min>
    </int>
    <int>
      <name>TrackDepthUncertainty</name>
      <max>63</max>
      <min>0</min>
    </int>
    <float algorithm="angle_0_360">
      <name>TrackHeading</name>
      <max>360</max>
      <min>0</min>
      <precision>1</precision>
    </float>
    <float>
      <name>TrackHeadingUncertainty</name>
      <max>10</max>
      <min>0</min>
      <precision>1</precision>
    </float>
    <float>
      <name>TrackSpeed</name>
      <max>12.8</max>
      <min>0</min>
      <precision>1</precision>
    </float>
    <float>
      <name>TrackSpeedUncertainty</name>
      <max>3</max>
      <min>0</min>
      <precision>1</precision>
    </float>
    <int>
      <name>DepthClassification</name>
      <max>7</max>
      <min>0</min>
    </int>
    <int>
      <name>TrackClassification</name>
      <max>7</max>
      <min>0</min>
    </int>
    <int>
      <name>TrackSpectralLevel1</name>
      <max>127</max>
      <min>0</min>
    </int>
    <int>
      <name>TrackFrequency1</name>
      <max>4095</max>
      <min>0</min>
    </int>
    <int>
      <name>TrackBandwidth1</name>
      <max>20</max>
      <min>0</min>
```

```
  </int>
  <int>
    <name>TrackSpectralLevel2</name>
    <max>127</max>
    <min>0</min>
  </int>
  <int>
    <name>TrackFrequency2</name>
    <max>4095</max>
    <min>0</min>
  </int>
  <int>
    <name>TrackBandwidth2</name>
    <max>20</max>
    <min>0</min>
  </int>
</layout>
<!-- decoding -->
<publish>
  <moos_var>NAFCON_MESSAGES</moos_var>
  <all />
</publish>
<!-- start pTransponderAIS replacement -->
<publish>
  <moos_var>AIS_REPORT</moos_var>
  <format>
    NAME=TRK_%1%_%2%,TYPE=track,UTC_TIME=%3$.0lf,
    X=%4%,Y=%5%,LAT=%6$lf,LON=%7$lf,
    SPD=%8%,HDG=%9%,DEPTH=%10%</format>
  <message_var>TrackNumber</message_var>
  <message_var algorithm="modem_id2name,to_lower">
    SourcePlatformId</message_var>
  <message_var>TrackTimestamp</message_var>
  <message_var algorithm="lon2utm_x:TrackLatitude">
    TrackLongitude</message_var>
  <message_var algorithm="lat2utm_y:TrackLongitude">
    TrackLatitude</message_var>
  <message_var>TrackLatitude</message_var>
  <message_var>TrackLongitude</message_var>
  <message_var>TrackSpeed</message_var>
  <message_var>TrackHeading</message_var>
  <message_var>TrackDepth</message_var>
</publish>
<publish>
  <moos_var type="double">
    TRK_%1%_%2%_NAV_UTC
  </moos_var>
  <format>%3$lf</format>
  <message_var>TrackNumber</message_var>
  <message_var algorithm="modem_id2name,to_upper">
    SourcePlatformId</message_var>
  <message_var>TrackTimestamp</message_var>
</publish>
<publish>
  <moos_var type="double">
    TRK_%1%_%2%_NAV_X
  </moos_var>
  <format>%3%</format>
  <message_var>TrackNumber</message_var>
  <message_var algorithm="modem_id2name,to_upper">
    SourcePlatformId</message_var>
  <message_var algorithm="lon2utm_x:TrackLatitude">
    TrackLongitude</message_var>
</publish>
<publish>
  <moos_var type="double">TRK_%1%_%2%_NAV_Y</moos_var>
  <format>%3%</format>
  <message_var>TrackNumber</message_var>
  <message_var algorithm="modem_id2name,to_upper">
   SourcePlatformId</message_var>
  <message_var algorithm="lat2utm_y:TrackLongitude">
          TrackLatitude</message_var>
</publish>
<publish>
  <moos_var type="double">TRK_%1%_%2%_NAV_LAT</moos_var>
  <format>%3$lf</format>
  <message_var>TrackNumber</message_var>
  <message_var algorithm="modem_id2name,to_upper">
    SourcePlatformId</message_var>
  <message_var>TrackLatitude</message_var>
</publish>
<publish>
  <moos_var type="double">TRK_%1%_%2%_NAV_LONG</moos_var>
  <format>%3$lf</format>
  <message_var>TrackNumber</message_var>
  <message_var algorithm="modem_id2name,to_upper">
    SourcePlatformId</message_var>
  <message_var>TrackLongitude</message_var>
</publish>
<publish>
  <moos_var type="double">TRK_%1%_%2%_NAV_SPEED</moos_var>
  <format>%3%</format>
  <message_var>TrackNumber</message_var>
  <message_var algorithm="modem_id2name,to_upper">
    SourcePlatformId</message_var>
  <message_var>TrackSpeed</message_var>
</publish>
<publish>
  <moos_var type="double">TRK_%1%_%2%_NAV_HEADING</moos_var>
  <format>%3%</format>
  <message_var>TrackNumber</message_var>
  <message_var algorithm="modem_id2name,to_upper">
    SourcePlatformId</message_var>
  <message_var>TrackHeading</message_var>
</publish>
<publish>
  <moos_var type="double">TRK_%1%_%2%_NAV_DEPTH</moos_var>
  <format>%3%</format>
  <message_var>TrackNumber</message_var>
  <message_var algorithm="modem_id2name,to_upper">
    SourcePlatformId</message_var>
  <message_var>TrackDepth</message_var>
</publish>
<!-- end pTransponderAIS replacement -->
</message>
...
```

Listing A.9 pGeneralCodex XML Configuration file `nafcon_targetsim.xml` *for transmitting simulated target state to all network nodes synchronously.*

```
<?xml version="1.0" encoding="ASCII" standalone="yes"?>
<!-- codec for broadcasting target state for simulators
 running on nodes -->
<message_set>
  <message>
    <name>SIM_TARGET</name>
    <outgoing_hex_moos_var>
      OUT_SIM_TGT_HEX_30B
    </outgoing_hex_moos_var>
    <incoming_hex_moos_var>
      IN_SIM_TGT_HEX_30B
    </incoming_hex_moos_var>
    <trigger>publish</trigger> <!-- publish to moos var -->
    <trigger_moos_var mandatory_content="tgt_delay=">
      TGT_STATE_OUT
    </trigger_moos_var>
    <size>30</size>
    <layout>
      <int>
        <name>tgt_x</name>
        <precision>1</precision>
        <max>100000</max>
        <min>-100000</min>
      </int>
      <int>
        <name>tgt_y</name>
        <precision>1</precision>
        <max>100000</max>
        <min>-100000</min>
      </int>
      <float>
        <name>tgt_depth</name>
        <precision>1</precision>
        <max>1000</max>
        <min>0</min>
      </float>
      <int>
        <name>tgt_hdg</name>
        <max>359</max>
        <min>0</min>
      </int>
      <float>
        <name>tgt_speed</name>
        <precision>1</precision>
        <max>30</max>
        <min>0</min>
      </float>
      <int>
        <name>tgt_freq</name>
        <max>5000</max>
        <min>0</min>
      </int>
      <int>
        <name>tgt_bw</name>
        <max>1000</max>
        <min>0</min>
      </int>
      <int>
        <name>tgt_spl</name>
        <max>200</max>
        <min>0</min>
      </int>
      <int>
        <name>tgt_delay</name>
        <max>5000</max>
        <min>0</min>
      </int>
      <int>
        <name>tn1_freq</name>
         <max>1000</max>
        <min>0</min>
      </int>
      <int>
        <name>tn1_dbl</name>
        <max>200</max>
        <min>0</min>
      </int>
      <int>
        <name>tn2_freq</name>
        <max>1000</max>
        <min>0</min>
      </int>
      <int>
        <name>tn2_dbl</name>
        <max>200</max>
        <min>0</min>
      </int>
      <int>
        <name>tn3_freq</name>
        <max>1000</max>
        <min>0</min>
      </int>
      <int>
        <name>tn3_dbl</name>
        <max>200</max>
        <min>0</min>
      </int>
      <int>
        <name>tn4_freq</name>
        <max>1000</max>
        <min>0</min>
      </int>
      <int>
        <name>tn4_dbl</name>
        <max>200</max>
        <min>0</min>
      </int>
      <int>
        <name>tn5_freq</name>
        <max>1000</max>
        <min>0</min>
      </int>
      <int>
        <name>tn5_dbl</name>
        <max>200</max>
        <min>0</min>
      </int>
      <int>
        <name>time</name>
        <max>2000000000</max>
```

```
        <min>1000000000</min>
      </int>
      <int>
        <name>src_num</name>
        <max>31</max>
        <min>0</min>
      </int>
    </layout>
    <!-- decoding -->
    <publish>
      <moos_var>TGT_STATE_IN</moos_var>
      <all />
    </publish>
    <publish>
      <moos_var>TARGET_CONTROL</moos_var>
      <format>ON</format>
    </publish>
  </message>
</message_set>
```