

# pMarineViewer: A GUI for Mission Monitoring and Control

December 2022

Michael Benjamin, mikerb@mit.edu  
Department of Mechanical Engineering  
MIT, Cambridge MA 02139  
project-pavlab/apps/app-pmviewer

---

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>Overviewx</b>   | <b>2</b>  |
| 1.1      | The Shoreside-Vehicle Topology . . . . .                               | 3         |
| 1.2      | Description of the pMarineViewer GUI Interface . . . . .               | 5         |
| 1.3      | The AppCasting, FullScreen and Traditional Display Modes . . . . .     | 6         |
| 1.4      | Run-Time and Mission Configuration . . . . .                           | 7         |
| 1.5      | Recent Changes and Bug Fixes . . . . .                                 | 8         |
| 1.5.1    | Release 22.8.x . . . . .   | 8         |
| 1.5.2    | Release 22.8 (Aug 2022) . . . . .                                      | 8         |
| <b>2</b> | <b>Command-and-Control</b>   | <b>9</b>  |
| 2.1      | Configurable Pull-Down Menu Actions . . . . .                          | 9         |
| 2.2      | Contextual Mouse Poking with Embedded OpArea Information . . . . .     | 9         |
| 2.3      | Action Button Configuration . . . . .                                  | 9         |
| 2.3.1    | Commander Pop-Up Window . . . . .                                      | 10        |
| <b>3</b> | <b>The BackView Pull-Down Menu</b>                                     | <b>11</b> |
| 3.1      | Panning and Zooming . . . . .  | 11        |
| 3.2      | Background Images . . . . .  | 12        |
| 3.3      | Local Grid Hash Marks . . . . .  | 14        |
| 3.4      | Full-Screen Mode . . . . .   | 14        |
| <b>4</b> | <b>Background Region Images</b>  | <b>14</b> |
| 4.1      | Default Packaged Images . . . . .                                      | 15        |
| 4.2      | Image File Format and Meta Data (Info Files) . . . . .                 | 15        |
| 4.3      | Obtaining Image Files . . . . .  | 16        |
| 4.4      | Loading Images at Run Time . . . . .                                   | 16        |
| 4.5      | Automatic alogview Detection of Background Image . . . . .             | 16        |
| 4.6      | Background Image Path . . . . .  | 17        |
| 4.7      | Troubleshooting . . . . .  | 18        |
| 4.7.1    | pMarineViewer fails to load the image (see only gray screen) . . . . . | 18        |
| 4.7.2    | pMarineViewer or alogview image is fine but no vehicles . . . . .      | 18        |
| 4.7.3    | alogview fails to load the image (see only gray screen) . . . . .      | 19        |
| <b>5</b> | <b>The GeoAttributes Pull-Down Menu</b>                                | <b>19</b> |
| 5.1      | Polygons, SegLists, Points, Circles and Vectors . . . . .              | 21        |
| 5.2      | Markers . . . . .  | 22        |
| 5.3      | Comms Pulses . . . . .   | 23        |
| 5.4      | Range Pulses . . . . .   | 25        |
| 5.5      | Drop Points . . . . .  | 26        |

|           |   |           |
|-----------|---|-----------|
| <b>6</b>  | <b>The Vehicles Pull-Down Menu</b>  | <b>28</b> |
| 6.1       | The Vehicle Name Mode . . . . .   | 28        |
| 6.2       | Dealing with Stale Vehicles . . . . .   | 29        |
| 6.3       | Supported Vehicle Shapes . . . . .  | 29        |
| 6.4       | Vehicle Colors . . . . .  | 31        |
| 6.5       | Centering the Image According to Vehicle Positions . . . . .                    | 31        |
| 6.6       | Vehicle Trails . . . . .  | 31        |
| <b>7</b>  | <b>The InfoCasting Pull-Down Menu</b>   | <b>32</b> |
| 7.1       | Turning On and Off InfoCast Viewing . . . . .                                   | 32        |
| 7.2       | Adjusting the InfoCast Viewing Panes Height and Width . . . . .                 | 33        |
| 7.3       | Adjusting the InfoCast Refresh Mode . . . . .                                   | 33        |
| 7.4       | Adjusting the InfoCast Fonts . . . . .  | 34        |
| 7.5       | AppCasting Versus RealmCasting . . . . .  | 35        |
| 7.6       | Adjusting the RealmCast Content . . . . .                                       | 36        |
| 7.7       | Additional RealmCast Capability: Watch Clusters . . . . .                       | 37        |
| 7.8       | Adjusting the AppCast and RealmCast Color Scheme . . . . .                      | 39        |
| <b>8</b>  | <b>The MOOS-Scope Pull-Down Menu</b>  | <b>39</b> |
| <b>9</b>  | <b>The Exclusion Filter</b>   | <b>40</b> |
| <b>10</b> | <b>The Action Pull-Down Menu</b>  | <b>41</b> |
| <b>11</b> | <b>The Mouse-Context Pull-Down Menu</b>   | <b>42</b> |
| 11.1      | Generic Poking of the MOOSDB with the Operation Area Position . . . . .         | 42        |
| 11.2      | Custom Poking of the MOOSDB with the Operation Area Position . . . . .          | 43        |
| <b>12</b> | <b>Configuring and Using the Commander Pop-Up Window</b>                        | <b>45</b> |
| 12.1      | Commander Pop-Up Window Actions and Content . . . . .                           | 46        |
| 12.2      | Commander Pop-Up Configuration . . . . .  | 47        |
| 12.3      | Commander Pop-Up Example Configuration from m2.bertha Mission . . . . .         | 48        |
| 12.4      | Commander Pop-Up Coordination with pShare and uFldShoreBroker . . . . .         | 48        |
| <b>13</b> | <b>Configuration Parameters for pMarineViewer</b>                               | <b>49</b> |
| 13.1      | Configuration Parameters for the BackView Menu . . . . .                        | 49        |
| 13.2      | Configuration Parameters for the GeoAttributes Menu . . . . .                   | 50        |
| 13.3      | Configuration Parameters for the Vehicles Menu . . . . .                        | 51        |
| 13.4      | Configuration Parameters for the InfoCasting Menu . . . . .                     | 52        |
| 13.5      | Configuration Parameters for the Scope, MouseContext and Action Menus . . . . . | 54        |
| 13.6      | Configuration Parameters for Optimizing in Extreme Load Situations . . . . .    | 54        |
| <b>14</b> | <b>Publications and Subscriptions for pMarineViewer</b>                         | <b>55</b> |
| 14.1      | Variables Published by pMarineViewer . . . . .                                  | 55        |
| 14.2      | Variables Subscribed for by pMarineViewer . . . . .                             | 56        |

---

## 1 Overviewx

The `pMarineViewer` application is a MOOS application written with FLTK and OpenGL for rendering vehicles and associated information and history during operation or simulation. A screen shot of a

simple one-vehicle mission is shown below in Figure 1.

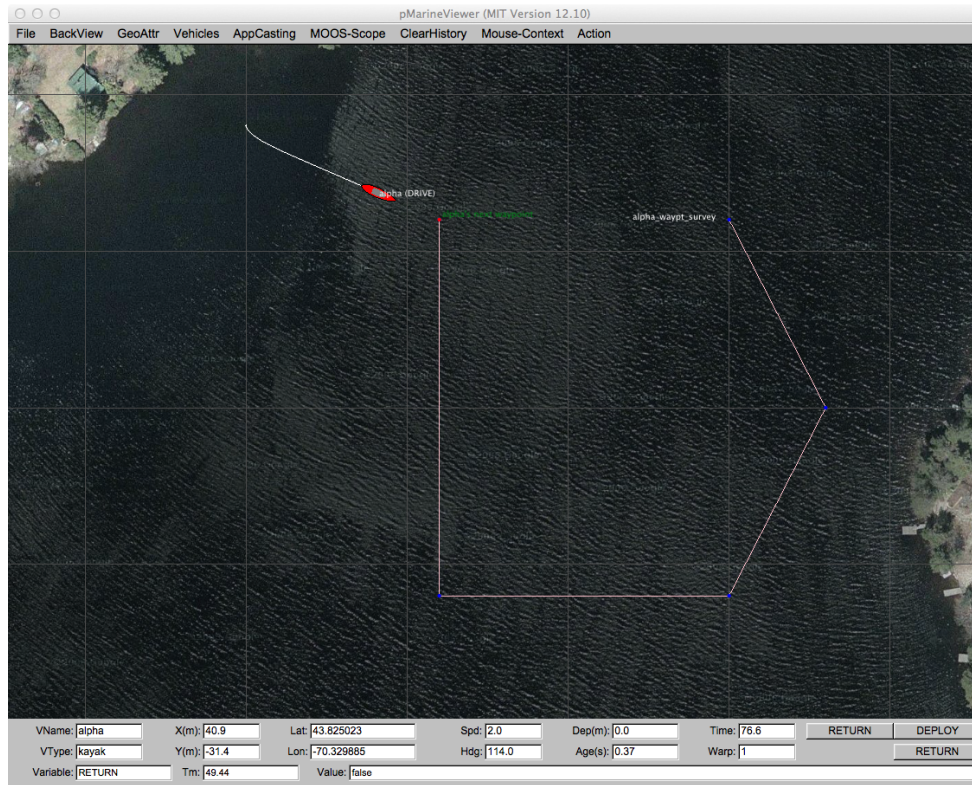


Figure 1: A `pMarineViewer` screen-shot executing a simple one-vehicle mission. The track of the vehicle is shown along with the set of waypoints it will traverse during this mission.

The user is able to manipulate a geo display to see multiple vehicle tracks and monitor key information about individual vehicles. In the primary interface mode the user is a passive observer, only able to manipulate what it sees and not able to initiate communications to the vehicles. However there are hooks available and described later in this section to allow the interface to accept field control commands. With Release 12.11, appcasting viewing is supported to allow the `pMarineViewer` user to view appcasts across multiple fielded vehicles within a single optional window pane. In the release following 19.8, appcasting was complemented with realmcasting allowing the user to view publications and subscriptions for any app on multiple fielded vehicles. The user may also set up a watch cluster of variables to monitor over a set of vehicles in a single pane. The realmcasting utilities add quite a bit of power to developers of multi-vehicle, i.e., swarm, autonomy missions. This is described more fully in Section 7.

## 1.1 The Shoreside-Vehicle Topology

In some simple simulation single-vehicle arrangements `pMarineViewer` may co-exist in the same MOOS community as the helm and other components of a simulated vehicle. This is the case in the Alpha example mission. A more typical module topology, however, is that shown in Figure 2, where `pMarineViewer` is running in its own dedicated local MOOS community while simulated vehicles, or

real vehicles on the water, transmit information in the form of a stream of *node reports* to the local community.

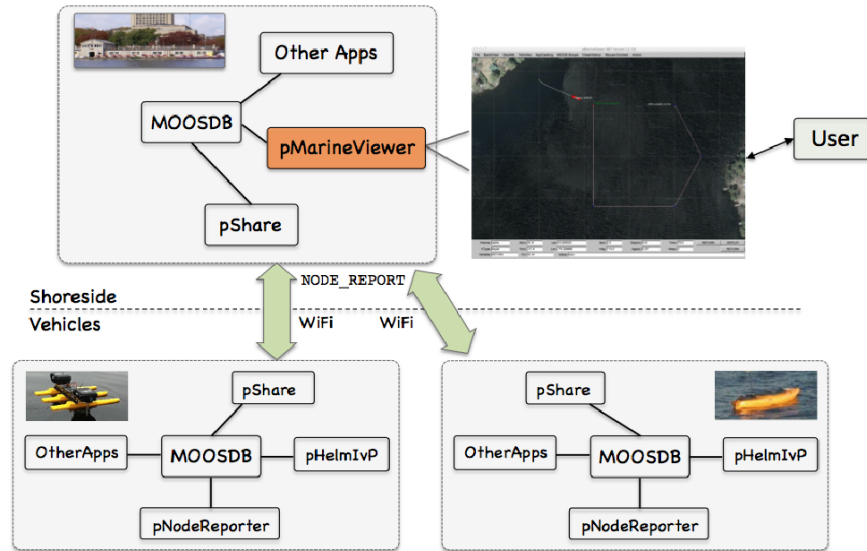


Figure 2: A common usage of the `pMarineViewer` is to have it running in a local `MOOSDB` community while receiving node reports on vehicle poise from other `MOOS` communities running on either real or simulated vehicles. The vehicles can also send messages with certain geometric information such as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

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

```
NODE_REPORT = "NAME=henry,TYPE=uuv,TIME=1195844687.236,X=37.49,Y=-47.36,SPD=2.40,
HDG=11.17,LAT=43.82507169,LON=-70.33005531,TYPE=KAYAK,MODE=DRIVE,
ALLSTOP=clear,index=36,DEP=0,LENGTH=4"
```

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

In addition to node reports, `pMarineViewer` subscribes to several other types of information typically originating in the individual vehicle communities. This include several types of geometric shapes for which `pMarineViewer` has been written to handle. This includes points, polygons, lists of line segments, grids and so on. This is described further in Section 5.

In addition to consuming the above information, `pMarineViewer` may also be configured to post certain information, usually for command and control purposes. Since this is mission-specific, this information is completely configured by the user to suit the mission. Posted information may also be tied to mouse clicks to allow, for example, a vehicle to be deployed to a point clicked by the users. This is described further in Section 2.

## 1.2 Description of the pMarineViewer GUI Interface

The viewable area of the GUI has three parts as shown in Figure 3 below. In the upper right, there is a geo display area where vehicles and perhaps other objects are rendered. The blue panes on the upper left displays appcast information. These panes hold appcast output from any appcast-enabled MOOS application running on any node, including the shoreside node. This is a new feature of Release 12.11 and may be toggled off and on with the 'a' key, and may be configured to be either open or closed by setting the `appcast_viewable` parameter inside the `pMarineViewer` MOOS configuration block.

In the lower pane, certain data fields associated with the *active* vehicle are updated. Multiple vehicles may be rendered simultaneously, but only one vehicle, the *active* will be reflected in the data fields in the lower pane. Changing the designation of which vehicle is active can be accomplished by repeatedly hitting the 'v' key. The active vehicle is always rendered as red, while the non-active vehicles have a default color of yellow. Individual vehicle colors can be given different default values (even red, which could be confusing) by the user.

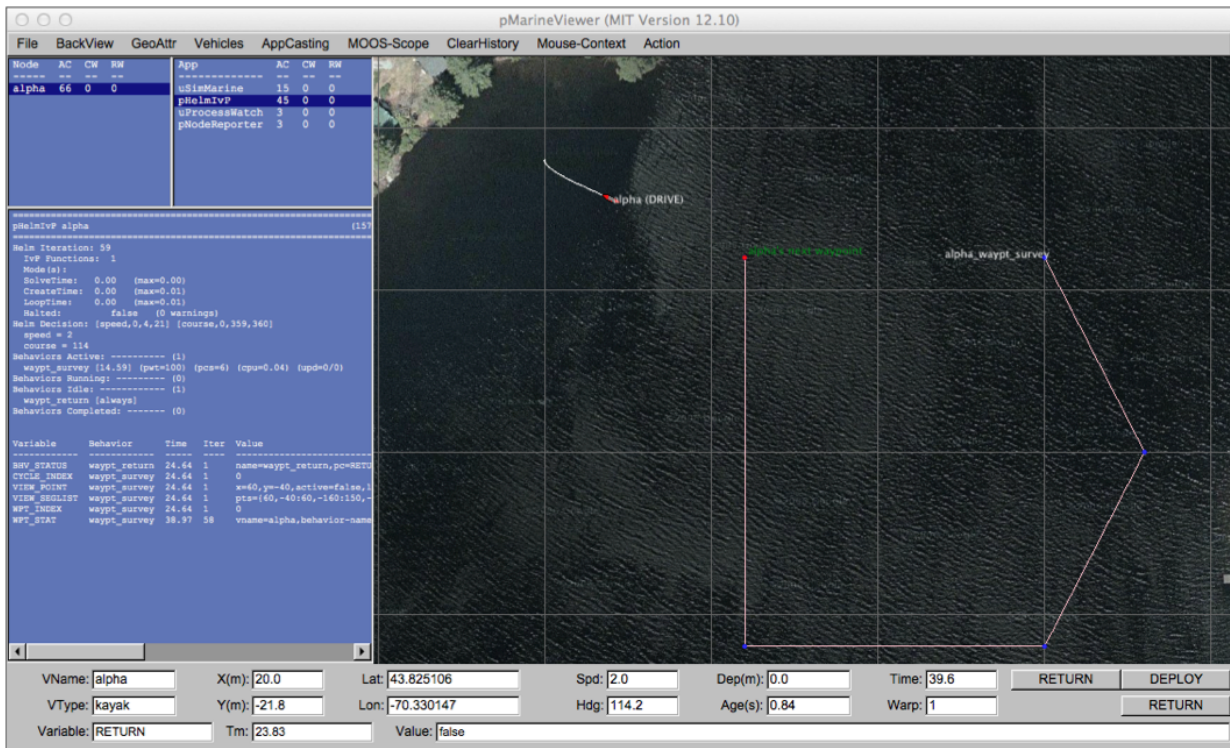


Figure 3: A screen shot of the pMarineViewer application running the alpha example mission. The position, heading, speed and other information related to the vehicle is reflected in the data fields at the bottom of the viewer.

Properties of the vehicle rendering such as the trail length, size, and color, and vehicle size and color, and pan and zoom can be adjusted dynamically in the GUI. They can also be set in the `pMarineViewer` MOOS configuration block. Both methods of tuning the rendering parameters are described later in this section. The individual fields of the data section are described below:

- **VName:** The name of the active vehicle associated with the data in the other GUI data fields. The

active vehicle is typically indicated also by changing to the color red on the geo display.

- **VType:** The platform type, e.g., `auv`, `uuv`, `glider`, `kayak`, `ship`, `heron`, `wamv`, `buoy`, `mokai`, `longship`, `swimmer` or `unknown`. The buoy shape is just a circle. The swimmer shape is a triangle.
- **X(m):** The x (horizontal) position of the active vehicle given in meters in the local coordinate system.
- **Y(m):** The y (vertical) position of the active vehicle given in meters in the local coordinate system.
- **Lat:** The latitude (vertical) position of the active vehicle given in decimal latitude coordinates.
- **Lon:** The longitude (horizontal) position of the active vehicle given in decimal longitude coordinates.
- **Spd:** The speed of the active vehicle given in meters per second.
- **Hdg:** The heading of the active vehicle given in degrees (0 – 359.99).
- **Dep(m):** The depth of the active vehicle given in meters.
- **Age(s):** The elapsed time in seconds since the last received node report for the active vehicle.
- **Time:** Time in seconds since `pMarineViewer` was launched.
- **Warp:** The MOOS Time-Warp value. Simulations may run faster than real-time by this warp factor. `MOOSTimeWarp` is set as a global configuration parameter in the `.moos` file.

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

The lower three fields of the window are used for scoping on a single MOOS variable. See Section 8 for information on how to configure the `pMarineViewer` to scope on any number of MOOS variables and select a single variable via an optional pull-down menu. The scope fields are:

- **Variable:** The variable name of the MOOS variable currently being scoped, or "n/a" if no scope variables are configured.
- **Time:** The variable name of the MOOS variable currently being scoped, or "n/a" if no scope variables are configured.
- **Value:** The actual current value for the presently scoped variable.

### 1.3 The AppCasting, FullScreen and Traditional Display Modes

As mentioned above, appcasting is new to release 12.11, `pMarineViewer` supports three display modes. The first mode is the *normal* mode familiar to pre-12.11 users of `pMarineViewer` as it was the only mode. A second mode, the *appcasting* mode, also shows the three appcasting panes shown above in Figure 3. The third mode is the *full-screen* mode which shows only the geo-display part to maximize viewing of the operation area. The modes may be toggled by single hot-key actions as shown in the figure.

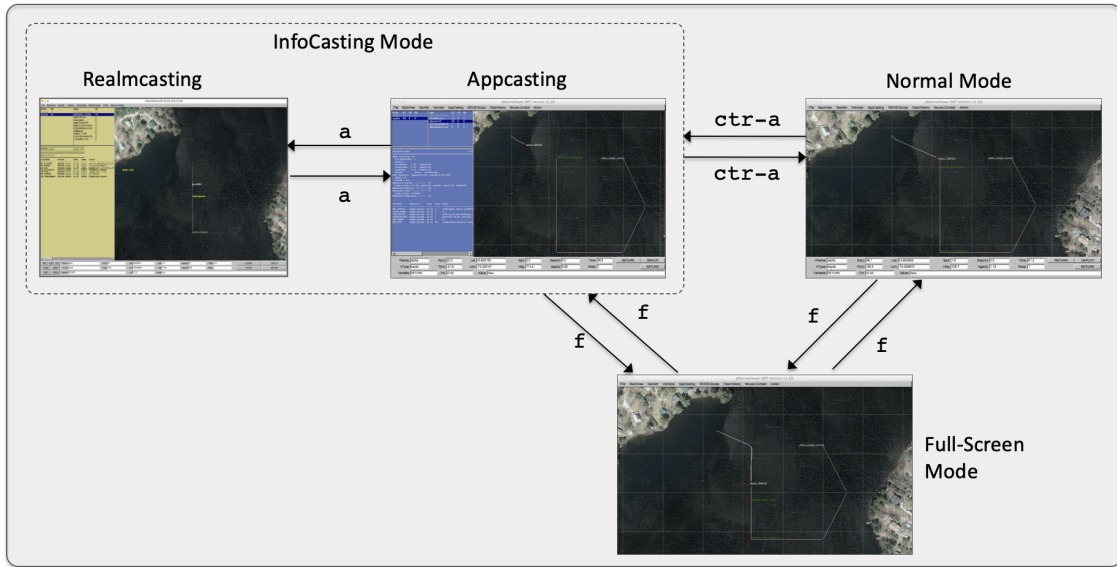


Figure 4: Three viewing modes are supported by pMarineViewer. The *normal* mode, the *infocasting* mode which renders appcast or realmcast output from any connected vehicle, or the *full-screen* mode to maximize viewing of the operation area and vehicles. The modes may be toggled with the hot-keys shown. When typing 'f' in the full-screen mode, the viewer will return to the mode prior to entering the full-screen mode. The modes may also be changed via pull-down menu items, or set to personal preferences in the .moos configuration block. In software released after 19.8.x, the infocasting mode has two sub-modes, appcasting and realmcasting.

To launch a mission in the appcasting mode, set `appcast_viewable=true` in the pMarineViewer configuration block. To launch in the full-screen mode, set `full_screen=true` in the configuration block instead.

### 1.4 Run-Time and Mission Configuration

Nearly all pMarineViewer configuration parameters may be configured both at run-time, via pull-down menu selections, and prior to launch via configuration lines in the pMarineViewer configuration block of the .moos mission configuration file. To reduce the need to consult the documentation, the text of the pull-down menu selection is identical to the text of the parameter in the configuration file. Furthermore, most parameter selections are a choice from a fixed set of options. The present option for a parameter is typically indicated by a radio button in the pull-down menu.

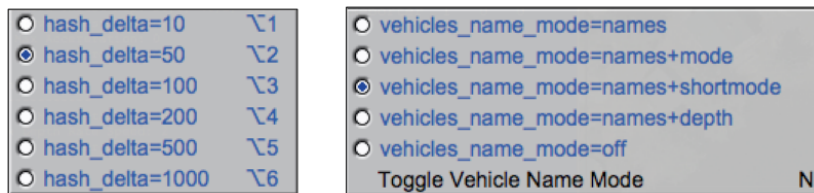


Figure 5: Most configuration parameters may be altered with pull-down menu selections. The radio-button shows the present parameter value and its neighbors show other legal settings. The text of the pull-down menu selection may be placed verbatim in the .moos configuration block to determine the setting upon the next mission launch. In general, menu items rendered in blue text are legally accepted parameters for placing in the .moos configuration block. Items in black are not.

Most parameter options have either a hot key associated with each option as shown in the left in Figure 5, or a hot key for toggling between options as on the right in the figure.

## 1.5 Recent Changes and Bug Fixes

### 1.5.1 Release 22.8.x

Changes in Release 22.8.x (the first release after 22.8, until 22.8.1 is released, this means trunk/main).

- A bug in loading multiple tiff files was fixed
- Loading of multiple tiff files is supported, no longer limited to just two.
- Tiff files can now be located using a shell path variable, `IVP_IMAGE_DIRS`.

### 1.5.2 Release 22.8 (Aug 2022)

- Major new augmentation to support RealmCasting, a powerful new tool for scoping on any app in a multi-vehicle mission. Clusters of variables can be configured to scope across multiple vehicles in a single table. This works in conjunction with a new app called pRealm, which requires no configuration and runs in each MOOS community. Toggling between appcasting and realmcasting is done with the 'a' key.
- Augmented the GUI to accept up to twenty buttons for command poking, up from the previous limit of four buttons. Buttons and info fields will automatically resize to accommodate however many buttons are used.
- Command buttons, when hovering with the mouse, will show what is being commanded upon a button click.
- `VPlug_GeoShapes` class was modified to explicitly drop from memory shapes that arrive with `active=false`. Previously this would just result in the object being ignored, but not removed. This mod guards against unbounded memory growth in pMarineViewer in some longer missions.
- `VPlug_AppCastSettings` in `lib-geometry` was replaced with `InfoCastSettings` in `lib-apputil`. This class stores all the user preferences applicable to appcasting and realmcasting. The new class is also used by `uMACView`.
- An additional variable, `REGION_INFO`, is published upon startup, and whenever a new vehicle has been detected. This info holds info about the background image, the zoom, the datum, and the pan x/y info. It is intended simply to be logged, and used by alogview upon startup to replicate the background image and orientation to be similar to how pMarineViewer was launched.
- Points and circles now have support to publish with a duration, and once the duration has been exceeded with now new publication (keyed on label), the object will be dropped from memory.
- Improved rapid drawing for large sets of Polygons
- Fixed bug where text/labels for objects off screen would be rendered on screen, appended to other objects' text, rather than simply not being drawn.
- New Option of ingesting `NODE_REPORT` info from `uFldNodeComms` as an intermediary. This enables smoother operation of pMarineViewer in missions with very high number of vehicles and very high time warp. Enabled with `node_report_unc=true` configuration. Of course must also be running `uFldNodeComms`.



## 2 Command-and-Control

For the most part `pMarineViewer` is intended to be only a receiver of information from the vehicles and the environment. Adding command and control capability, e.g., widgets to re-deploy or manipulate vehicle missions, can be readily done, but make the tool more specialized, bloated and less relevant to a general set of users. However, `pMarineViewer` does have a few powerful extendible command and control capabilities under the hood. Each are simply ways to conveniently post to the MOOSDB, and come in three forms: (a) configurable pull-down menu actions, and (b) contextual mouse poking with embedded oparea information, (c) configurable action buttons, and in Release 17.7 and later, (d) a configure commander pop-up window.

### 2.1 Configurable Pull-Down Menu Actions

The Action pull-down menu described in Section 10 provides a way to pre-define a set of MOOS postings, each selectable from the pull-down menu. For example, the alpha mission is configured with the below action:

```
action = RETURN = true
```

This post to the MOOSDB correlates to a behavior condition of the helm waypoint behavior with the return position. Actions may also be grouped into a single pull-down selection, discussed in Section 10.

### 2.2 Contextual Mouse Poking with Embedded OpArea Information

The mouse left and right buttons may be configured to make a post to the MOOSDB with value partly comprised of the point in the oparea under the mouse when clicked. For example, rather than commanding the vehicle to return to a pre-defined return position as the case above implies, the user may use this feature to command the vehicle to a point selected by the user at run time with a mouse click. The configuration might look like:

```
left_context[return] = RETURN_POINT = points = $(XPOS),$(YPOS)
left_context[return] = RETURN = true
```

This is discussed further in Section 11.

### 2.3 Action Button Configuration

Perhaps the most visible form of command and control is with the few action buttons configurable for on-screen use. For example, the DEPLOY and RETURN buttons in the lower right corner as in Figures 1, and 3. These buttons, for example, are configured as follows:

```
button_one = DEPLOY # DEPLOY=true
button_one = MOOS_MANUAL_OVERRIDE=false # RETURN=false
button_two = RETURN # RETURN=true
```

The general syntax is:

```
button_one   = <label> # <MOOSVar>=<value> # <MOOSVar>=<value> ...
button_two   = <label> # <MOOSVar>=<value> # <MOOSVar>=<value> ...
button_three = <label> # <MOOSVar>=<value> # <MOOSVar>=<value> ...
button_four  = <label> # <MOOSVar>=<value> # <MOOSVar>=<value> ...
```

The left-hand side contains one of the four button keywords, e.g., `button_one`. The right-hand side consists of a '#'-separated list. Each component in this list is either a '#'-separated variable-value pair, or otherwise it is interpreted as the button's label. The ordering does not matter and the '#'-separated list can be continued over multiple lines as in the simple example above.

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

The couple of macros are supported for button clicks:

- `$(BIX)`: An integer that increments on each button click, regardless of whether other button clicks contain this macro. It starts at zero.
- `$(UTC)`: The current time in UTC seconds, in millisecond precision.

If either of these macros appear in isolation, they will be published as type double, not type string. Support for this in the release after Release 19.8.1.

### 2.3.1 Commander Pop-Up Window

In Releases 17.7 the *commander pop-up* feature was added to `pMarineViewer`. This can be thought of as the tool to use when your mission requires the more than the four buttons allowed on the lower right corner of the main screen (the *Action Buttons* described above in Section 2.3. The commander pop-up window contains user-configured commands to either all vehicles, or particular vehicles, and commands to the shoreside if desired. If `pMarineViewer` is configured to use the commander pop-up, this window is toggled open/closed with the space-bar key. It also contains a log at the bottom of the window, showing exactly what pokes to the MOOSDB are made upon each button click.

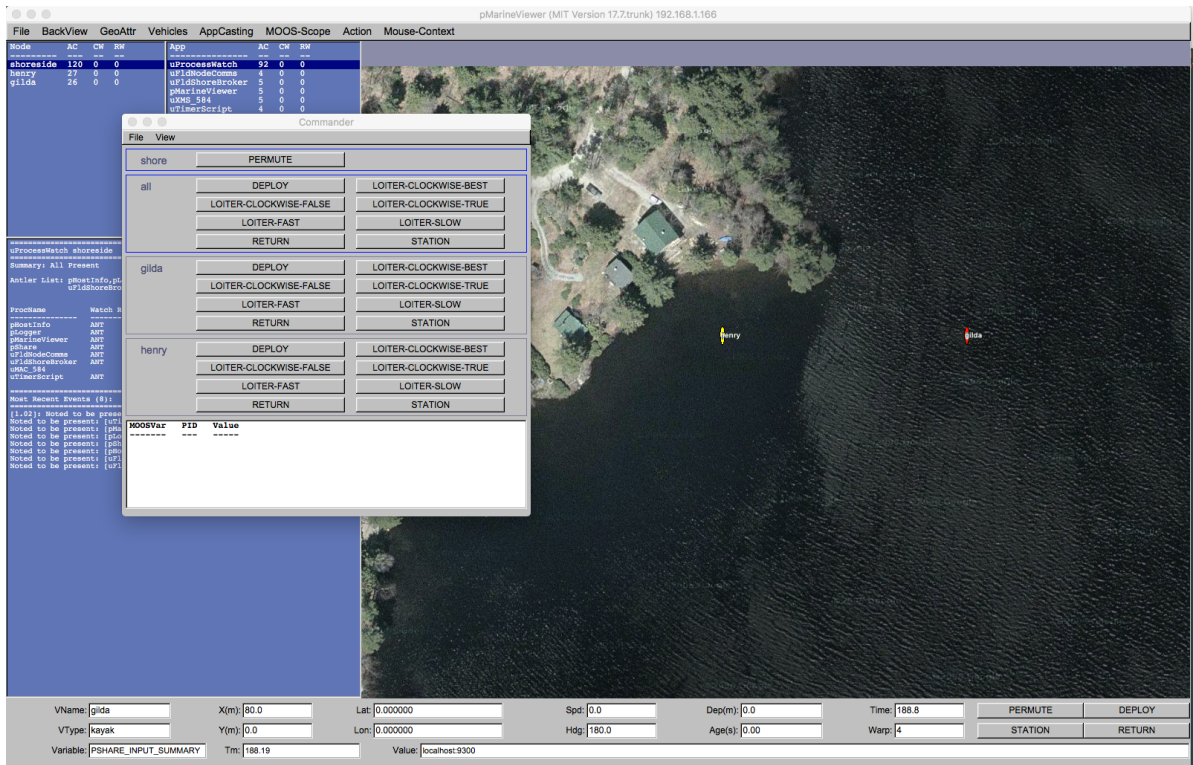


Figure 6: The commander pop-up window is toggled open/closed with the space-bar key and presents the user with a set of user-configured commands to either the all vehicles, individual vehicles, or the the pMarineViewer shoreside community itself.

The above figure was generated from the m2\_berta mission, which may be used as a good starting example. This pop-up window is configured in the pMarineViewer configuration block as described further in Section 12.

### 3 The BackView Pull-Down Menu

The BackView pull-down menu deals mostly with panning, zooming and issues related to the rendering of the background on which vehicles and mission artifacts are rendered. The full menu is shown in Figure 7. Although panning and zooming is not something typically done via the pull-down menu, they are included in this menu primarily to remind the user of their hot-keys. The zooming commands affect the viewable area and apparent size of the objects. Zoom in with the 'i' or 'I' key, and zoom out with the 'o' or 'O' key. Return to the original zoom with ctrl+'z'.

#### 3.1 Panning and Zooming

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

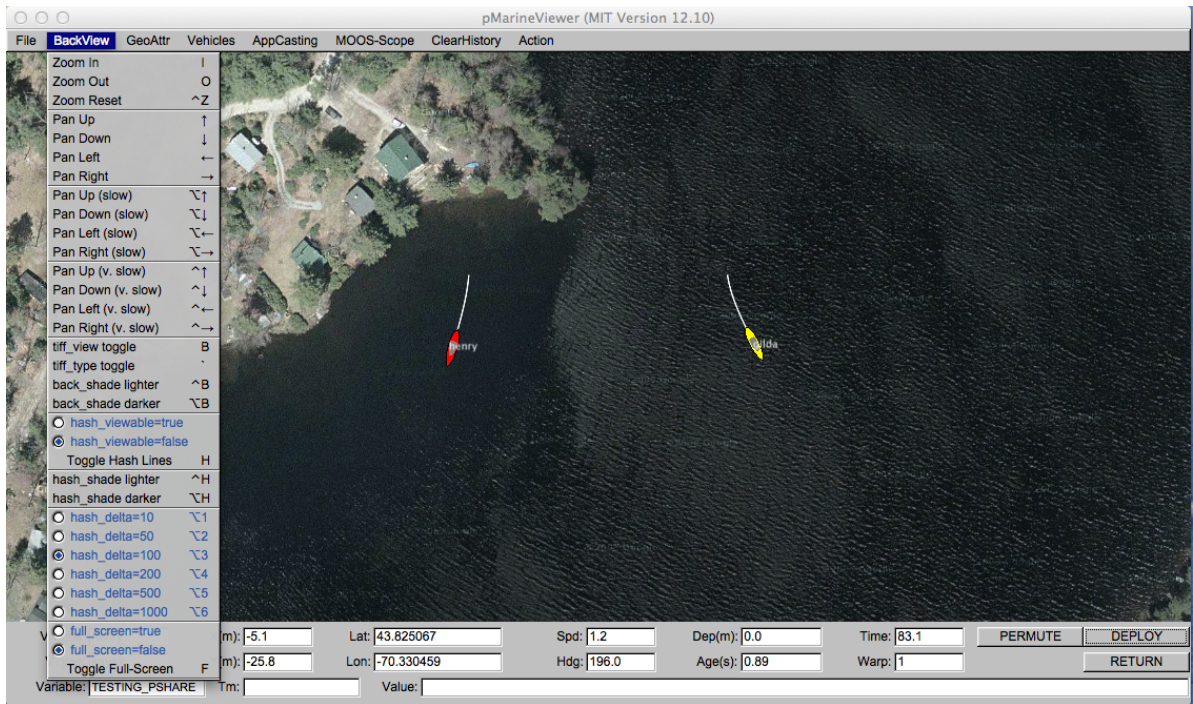


Figure 7: **The BackView menu:** This pull-down menu lists the options, with hot-keys, for affecting rendering aspects of the geo-display background.

positions at the center of the screen with the 'c' key. These are part of the 'Vehicles' pull-down menu discussed in Section 6.

### 3.2 Background Images

The background can be in one of two modes; either displaying a gray-scale background, or displaying a geo image read in as a texture into OpenGL from an image file. The default is the geo display mode if provided on start up, or the grey-scale mode if no image is provided. The mode can be toggled by typing the 'b' or 'B' key. The geo-display mode can have two sub-modes if two image files are provided on start-up. This is useful if the user has access to a satellite image *and* a map image for the same operation area. The two can be toggled by hitting the back tick key. After Release 22.8.x, multiple tiff files may be provided, and toggling will cycle through all tiff files.

When in the grey-scale mode, the background can be made lighter by hitting the ctrl+'b' key, and darker by hitting the alt+'b' key.

To use an image in the geo display, the input to `pMarineViewer` comes in two files, an image file in TIFF format, and an information text file correlating the image to the local coordinate system. The file names should be identical except for the suffix. For example `dabob_bay.tif` and `dabob_bay.info`. Only the `.tif` file is specified in the `pMarineViewer` configuration block of the MOOS file, and the application then looks for the corresponding `.info` file. The info file correlates the image to the local coordinate system and specifies the location of the local (0,0) point. An example is given in Listing 1.

*Listing 3.1: An example .info file associated with a background image.*

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

All four latitude/longitude parameters are mandatory. The two datum lines indicate where (0,0) in local coordinates is in earth coordinates. However, the datum used by `pMarineViewer` is determined by the `LatOrigin` and `LongOrigin` parameters set globally in the MOOS configuration file. The datum lines in the above information file are used by applications other than `pMarineViewer` that are not configured from a MOOS configuration file. The `lat_north` parameters correlate the upper edge of the image with its latitude position. Likewise for the other three parameters and boundaries. Two image files may be specified in the `pMarineViewer` configuration block. This allows a map-like image and a satellite-like image to be used interchangeably during use. An example of this is shown in Figure 8 with two images of Dabob Bay in Washington State.

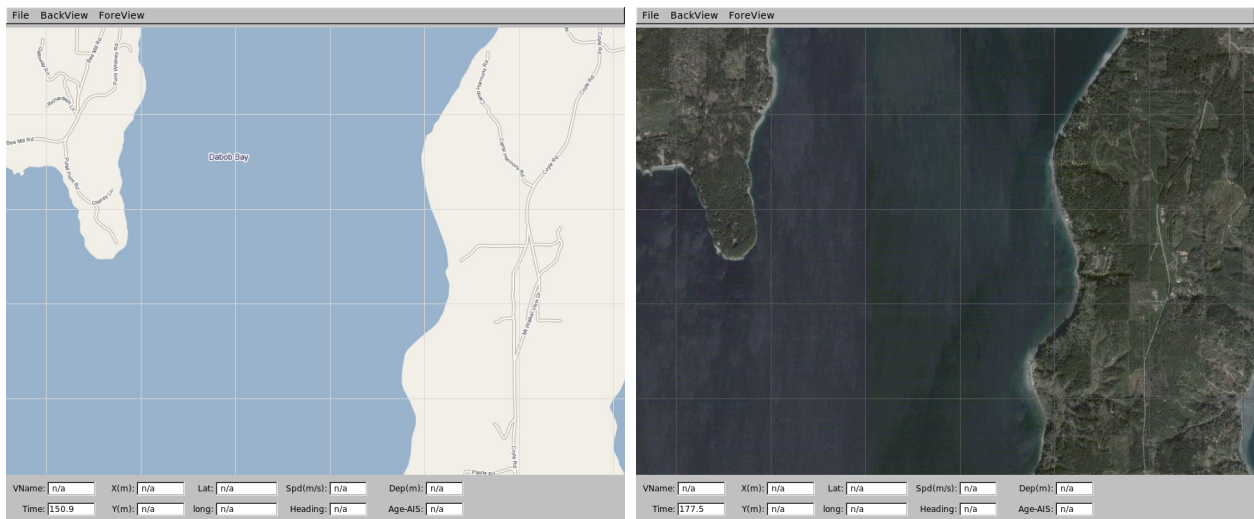


Figure 8: **Dual background geo images:** Two images loaded for use in the geo display mode of `pMarineViewer`. The user can toggle between both as desired during operation.

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

```
tiff_file = dabob_bay_map.tif
tiff_file_b = dabob_bay_sat.tif
```

In Release 22.8.x, more than two tiff files may be provided, with successive `tiff_file` lines. The parameter `tiff_file_b` has been deprecated. By default `pMarineViewer` will look for the files `Default.tif` and `DefaultB.tif` in the local directory unless alternatives are provided in the configuration block.

In Release 22.8.x `pMarineViewer` (and `alogview`) make use of a shell environment variable `IVP_IMAGE_DIRS`. This is a colon-separated path of directories on the local computer where images files are searched for upon `pMarineViewer` startup. Image files with paths relative to the mission directory are still supported. One directory, `moos-ivp/ivp/data/` is hard-coded into the `pMarineViewer` containing the Alpha mission image (Forest Lake Maine), and the MIT Sailing Pavilion.

By default a copies of the background image and info files are *not* logged by `pLogger`. This may be changed by the setting the following parameter: `log_the_image = true`. This is only a request to `pLogger` in the form of the `PLOGGER_CMD` posting:

```
PLOGGER_CMD = COPY_FILE_REQUEST = /home/jake/images/lake_george.tif
PLOGGER_CMD = COPY_FILE_REQUEST = /home/jake/images/lake_george.info
```

The result should be that the files are included in the folder created by `pLogger` with the `.tif` and `.info` suffixes. These may then be used by post-mission analysis tools to re-convey the operation area.

### 3.3 Local Grid Hash Marks

Hash marks can be overlaid onto the background. By default this mode is off, but can be toggled with the 'h' or 'H' key, or set in the configuration file with the `hash_viewable` parameter. The hash marks are drawn in a grey-scale which can be made lighter by typing the `ctrl+'h'` key, and darker by typing the `alt+'h'` key, or set in the configuration file with the `hash_shade` parameter. The hash mark spacing may only be set to one of the values shown in the menu. If set to different value, the closest legal value will be chosen.

### 3.4 Full-Screen Mode

The viewer may be put into full-screen mode by toggling with the 'f' key. This will result in the data fields at the bottom of the viewer being replace with a bit more viewing area for the geo display. As with all other blue items in this pull-down menu, the full-screen mode may be set in the MOOS configuration block with `full_screen=true`. The default if false. Full-screen mode is useful when running simulations while connected to a low-resolution projector for example.

## 4 Background Region Images

In both `pMarineViewer` and `alogview`, typical use involves a background region image upon which vehicles and other geometric objects are overlaid. Both utilies can be configured to use images of their own choosing. See examples in Figure 9. The user provides these images from whatever source they wish. Options for obtaining images and proper format are discussed below in the section *Obtaining Image Files*.



Figure 9: **Example Background Region Images:** The example images downloaded from freely available tile servers that may be utilized in either `pMarineViewer` or `alogview`.

## 4.1 Default Packaged Images

Image files can be large, and of course there are endless possibilities depending on where you are operating and which kind of background images you prefer. That being said, a couple images are distributed with MOOS-IvP. This allows example missions distributed with the MOOS-IvP code to having working images out-of-the-box without requiring the new user to fetch images. These two images are for (a) the MIT Sailing Pavilion, and (b) Forest Lake in Gray Maine, the site of some of the earliest in-water experiments circa 2004. This files are:

- `MIT_SP.tif`
- `forrest19.tif`

Both are distributed with MOOS-IvP and can be found in `moos-ivp/ivp/data`. When `pMarineViewer` or `alogview` is launched, this directory will be examined for the specified image file. Instructions for loading user-provided images is are given below in section *Loading Images at Run Time*.

## 4.2 Image File Format and Meta Data (Info Files)

Images loaded into `pMarineViewer` and `alogview` are in the format of "Tiff" files. These files have the suffix ".tif". Tiff files have been around since the eighties, use lossless compression, are typically high quality, but are not as common as formats such as jpeg. There are many freely available tools for converting back and forth between tiff and jpeg and other formats. Tiff files are used in `pMarineViewer` and `alogview` primarily due to the availability of the `libtiff` library, readily available through package managers on both the MacOS and Linux platforms.

Each .tif file used in `pMarineViewer` and `alogview` has a corresponding .info file, containing (a) the latitude and longitude coordinates of the image edges, and (b) the datum, or (0,0) coordinate, on the image. For example, the `MIT_SP.tif` file has a corresponding `MIT_SP.info` file found in the same directory.

```
lat_north = 42.360264
lat_south = 42.35415
lon_east  = -71.080058
lon_west  = -71.094214
datum_lat = 42.358436
datum_lon = -71.087448
```

### 4.3 Obtaining Image Files

Image files may be obtained and used in `pMarineViewer` and `alogview` from any source convenient to the user. This includes opening an image in say Google Maps on a web browser and performing a screen grab. The drawback of this method however is that it may be hard to precisely determine the lat/lon coordinates of the edges used in the corresponding `.info` file.

There are several open *tile servers* which allow a user to download an image tile, or set of tiles, provided with a range of lat/lon coordinates. These tiles can then be stitched together to make a single image. Although this sounds cumbersome, this process can be automated in a script. One such script is Anaxi Map, written by Conlan Cesar:

```
https://github.com/HeroCC/AnaxiMap
```

This utility is capable of using one of several tile servers with various background styles, such as Google Maps, maps with terrain or bathymetry information, or maps with street data. See Figure 9.

AnaxiMap, or similar utilities, may produce images in jpeg or png format. MacOS and Linux provide native utilities for converting the format, or "exporting" the file, to tiff format. On MacOS or Linux, if the free ImageMagick package is installed, you can use the "convert" utility:

```
$ convert region.jpg region.tif
```

### 4.4 Loading Images at Run Time

Post Release 22.8, both `pMarineViewer` and `alogview` support operation with multiple background images. Toggling between images at run time is done by selecting the `BackView` pull-down menu and selecting `tiff_type toggle`, or by simply hitting the ' (back-tick) key.

In `pMarineViewer`, the multiple background images may be specified with multiple configuration lines, for example:

```
tiff_file = MIT_SP.tif  
tiff_file = mit_sp_osm18.tif
```

In `alogview`, the multiple background images may be specified on the command line:

```
$ alogview --bg=MIT_SP.tif --bg=mit_sp_osm18.tif file.alog
```

### 4.5 Automatic `alogview` Detection of Background Image

When launching `alogview` typically the user wants to use the same background region image used by `pMarineViewer` during the course of the mission that generated the `alog` file. In a new feature, post Release 22.8, `alogview` will automatically attempt to detect the image file used by `pMarineViewer`. The name of region image is now published by `pMarineViewer` during the execution of the mission. This information is contained in the variable `REGION_INFO`. For example:



```
REGION_INFO = lat_datum=42.358436, lon_datum=-71.087448, img_file=MIT_SP.tif,\
              zoom=2.5, pan_x=129, pan_y=-364
```

The region info contains the name of the image (tiff) file used during the course of the mission, as well as the pan and zoom information as hints for `alogview` for use upon startup. The images found in `REGION_INFO` in the `alog` file will be loaded, as well as any images specified on the command line with the `--bg` options.

## 4.6 Background Image Path

Support for the `IVP_IMAGE_DIRS` shell path is a new feature, post Release 22.8, relevant to both `pMarineViewer` and `alogview`. This is explained below.

Image files are named in either the `pMarineViewer` config block of the `.moos` mission file, or named on the command line when launching `alogview`, as specified in Section *Loading Images at Run Time*. For `alogview`, the file may also be named in the `REGION_INFO` logged variable as discussed above.

Both apps need to *find* the named `.tif` file. When found, it looks for the corresponding `.info` file in the same directory. There are four options for making this work:

- The image file is in the same directory as the mission file.
- The image file is in the special directory `moos-ivp/ivp/data`.
- The full or relative path name of the file is specified.
- The file exists in a directory on your `IVP_IMAGE_DIRS` path.

The first option has the drawback of duplicating the image file in potentially many places. The second option has the drawback that the directory `moos-ivp/ivp/data` is part of the MOOS-IvP code distribution which users otherwise consider to be read-only. A fresh check out of MOOS-IvP would reset this directory and users would need to take care to migrate files to a new checkout. The third option is that full or relative path name may not be the same between different users or machines. The fourth option is the newest option and arguably has the least downside.

The `IVP_IMAGE_DIRS` is shell (e.g., `bash`) environment variable. It contains a list of one or more directories on your local computer where `pMarineViewer` or `alogview` will look when attempting to load image files. Shell environment variables are already common settings that users will customize on their particular machine.

The recommended way for users to use a set of custom image files is to (a) organize them in one or more directories, preferably under version control, (b) install them at a convenient location on your local machine, (c) configure the `IVP_IMAGE_DIRS` shell variable to contain the one or more directories where your image files reside.

For example, if you have a folder of image files with the following structure:

```
my_images/  
  napa_bay/  
    napa_bay_gmaps.tif  
    napa_bay_gmaps.info  
    napa_bay_osm.tif  
    napa_bay_osm.info  
  happy_harbor/  
    happy_harbor_gmaps.tif  
    happy_harbor_gmaps.info  
    happy_harbor_osm.tif  
    happy_harbor_osm.info
```

If this folder is installed on your machine in the home directory folder call "project", then you would set your IVP\_IMAGE\_DIRS path in your shell configuration file, e.g., .bashrc, as follows:

```
IVP_IMAGE_DIRS=~ /project/napa_bay  
IVP_IMAGE_DIRS+=:~/project/happy_harbor
```

To verify which file has been loaded, the appcasing output of `pMarineViewer` shows the full path name of the loaded file(s). And when `alogview` is launched, the terminal output indicates which directories are being searched, in order, for the image files. This information may be obscured however when the `alogview` window pops up, but you can find it if you go back to it and perhaps scroll up. Note: It is not sufficient, in the example above, to simply set `IVP_IMG_DIR= /project`, the parent directory of all image folders. Each image folder must be named.

## 4.7 Troubleshooting

### 4.7.1 pMarineViewer fails to load the image (see only gray screen)

1. Check the appcasing output of `pMarineViewer`. The top few lines should show which image file is loaded. Is this a file you recognize?
2. Does this file exist on your computer? Verify it is where you think it is.
3. How are you specifying this file in your `pMarineViewer` config block? If it is specified as a relative path, e.g., `../my_images/napa_bay.tif`, make sure that relative path location is correct.
4. If you are specifying the image file with just the file name (no path information), then check you have your `IVP_IMAGE_DIRS` variable set properly. Run `echo $IVP_IMAGE_DIRS` on the command line.
5. Simplest but most common: Make sure your image file name (configuration and actual name) end in the suffix `.tif` and not `.tiff`.

### 4.7.2 pMarineViewer or alogview image is fine but no vehicles

1. Check the `.info` file. Make sure the lat/lon values sanity check, e.g., rough magnitude, relative values.
2. Make sure the `datum_lat` and `datum_lon` values are in the range of the image.
3. Make sure the `datum_lat` and `datum_lon` values match the datum set at the top of the vehicle and shoreside mission files.

### 4.7.3 alogview fails to load the image (see only gray screen)

In the newer version of `alogview`, when launching it will attempt to read the name of the image file used by `pMarineViewer`. So if `pMarineViewer` launched ok, chances are good `alogview` will also launch with the same image. However, it could be the case that (a) the mission was run on some other computer that contained the image file and your current computer does not. The image file is not logged. (b) the mission was named

1. Does this file exist on your computer? Verify it is where you think it is. It is possible that you are trying to run `alogview` on your computer with alog files generated on someone else's computer. Make sure you have the image file.
2. Check the terminal output of `alogview` as it is loading. To demonstrate the below output from an `alogview` launch purposely use the file `dforrest19.tif` instead of `forrest19.tif`. Note the sequences of folders searched in the attempt to find the image file. The first attempt is the in the `moos-ivp/ivp/data` directory. The next attempts are based on the value of `IVP_IMAGE_DIRS`. The final attempt is in the current working directory where `alogview` was launched. Does this match up with your expectations?

```
TIFF FILES COUNT:1
[1] Looking for dforrest19.tif and dforrest19.info in:
    Dir: [/Users/james/moos-ivp/ivp/data]
    Not found.
[2] Looking for dforrest19.tif and dforrest19.info in:
    Dir: [/Users/james/pavlab_map_images/popolopen]
[3] Looking for dforrest19.tif and dforrest19.info in:
    Dir: [/Users/james/pavlab_map_images/mit]
[4] Looking for dforrest19.tif and dforrest19.info in:
    Dir: [/Users/james/moos-ivp/ivp/datax]
[5] Looking for dforrest19.tif and dforrest19.info in:
    Dir: [/Users/james/moos-ivp/ivp/data-local]
[6] Looking for dforrest19.tif and dforrest19.info in:
    Dir: [/Users/james/moos-ivp/ivp/data]
[*] Looking for dforrest19.tif and dforrest19.info in:
    Dir: [./]
    Not found.
Could not find the pair of files:
dforrest19.tif and dforrest19.info
Opening Tiff:
TIFFOpen: : No such file or directory.
Failed!!!!!!!!!!
```

3. Simplest but most common: Make sure your image file name (configuration and actual name) end in the suffix `.tif` and not `.tiff`.

## 5 The GeoAttributes Pull-Down Menu

The GeoAttributes pull-down menu allows the user to affect viewing properties of geometric objects capable of being rendered by the `pMarineViewer`. The viewer subscribes for and supports the following geometric objects, typically generated by the helm or other MOOS applications:

- Polygons
- SegLists
- Points
- Vectors
- Circles
- Markers
- RangePulses
- CommsPulses

The viewer will also render the following other geometric objects set either in the configuration file or interactively by the user:

- Datum
- OpArea
- DropPoints

The Datum is simply the point in local coordinates representing (0,0). The pull-down menu allows the user to toggle off or on this rendering of the datum point as well as adjust its size and color. The OpArea is used to render the boundaries, if they exist, of an area of operation. DropPoints (described further in Section 5.5) are labeled points the user may drop on the viewing area for reference or mission planning

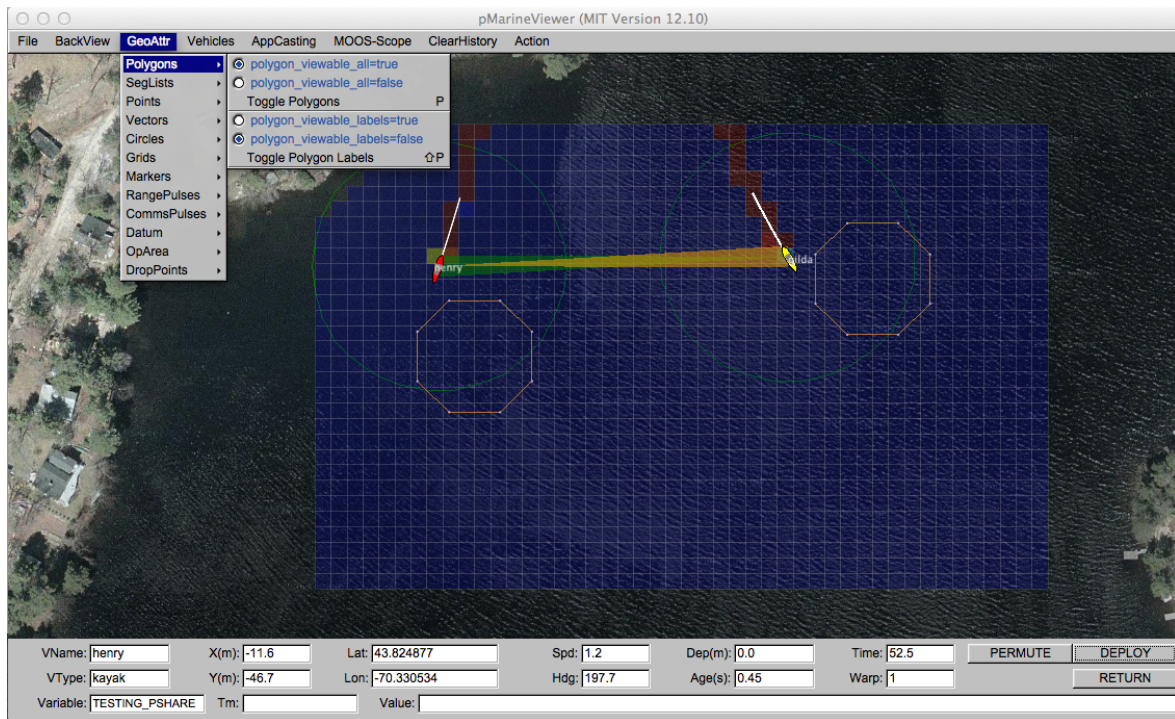


Figure 10: **The "GeoAttr" menu:** This pull-down menu lists the options and hot keys for affecting the rendering of geometric objects.

The possible parameters settings for rendering the geometric objects received by `pMarineViewer` via MOOS mail is provided in Section 13.2.

## 5.1 Polygons, SegLists, Points, Circles and Vectors

The five geometric objects, polygons, seglists, points, circles and vectors, provide a core rendering capability for applications (like the helm and its behaviors) to render visual artifacts related to the unfolding of a mission. For example, in Figure 1, a seglist is used to render the vehicle waypoints, and a labeled point is used to render the vehicles current next waypoint.

Objects are passed to `pMarineViewer` as strings via normal MOOS mail. An example is given below for the seglist shown in Figure 1. The string is a comma-separated list of variable=value pairs. Note the last pair is a label. Labels are used by all five object types to distinguish uniqueness.

```
VIEW_SEGLIST = pts={60,-40:60,-160:150,-160:180,-100:150,-40},label=waypt_survey
```

Uniqueness is used to either overwrite or erase previously rendered object instances. For example the above seglist could be "moved" five meters south by posting an identical message with the same label and adjusted coordinates. The *source* of the object is also tracked by `pMarineViewer`. This is given by the MOOS community from which the message originated, and typically represents the vehicle's name. Thus the above seglist could also be "moved" if the posting originated from a second vehicle community, in the type of arrangement shown in Figure 2.

### Parameters Common to Polygons, SegLists, Points, Circles and Vectors

Other optional parameters may be associated with an object to specify rendering preferences. They include:

- `active`
- `msg`
- `vertex_size:`
- `vertex_color`
- `edge_size`
- `edge_color`
- `fill_color`
- `fill-transparency`

For example, the `VIEW_SEGLIST` specification above may be augmented with the below string to specify edge and vertex size and color preferences:

```
edge_color=pink,vertex_color=blue,vertex_size=4,edge_size=1
```

The `active` parameter may be set to false to indicate that an object, previously received with the same label, should not be drawn by `pMarineViewer`. The `msg` parameter may be used to override

the string rendered as the object's label. Since labels are used to uniquely identify an object, the `msg` parameter may be used to, for example, draw five points all with same rendered text. The other six parameters are self-explanatory and not necessarily relevant to all objects. For example, `pMarineViewer` will ignore an `edge_size` specification when drawing a point, and a `fill_color` will only be relevant for a polygon and a circle.

## Serializing Geometric Objects for pMarineViewer Consumption

Geometric objects are only *consumed* by `pMarineViewer`, but it's worth discussing the issue of *generating* and serializing an object into a string. It is possible to simply post a string in the right format, as with:

```
string str = "x=5,y=25,label=home,vertex_size=3"; // Not recommended
m_Comms.Notify("VIEW_POINT", str);
```

It is highly recommended that this be left to the serialization function native to the C++ class.

```
#include "XYPoint.h"

XYPoint my_point(5, 25); // Recommended
my_point.set_label("home");
my_point.set_vertex_size(3);
string str = my_point.get_spec();
m_Comms.Notify("VIEW_POINT", str);
```

The latter code is less prone to user error, and is more likely to work in future code releases if the underlying formats need to be altered. (This is the idea behind Google Protocol Buffers, but here the geometric classes are implemented with various geometry function relations defined in addition to the serialization and de-serialization.) The full set of interface possibilities for creating and manipulating geometry objects is beyond the scope of the discussion here however.

## 5.2 Markers

A set of marker object types are defined for rendering characteristics of an operation area such as buoys, fixed sensors, hazards, or other things meaningful to a user. The six types of markers are shown in Figure 11. They are configured in the `pMarineViewer` configuration block of the MOOS file with the following format:

```
marker = type=efield,x=100,y=20,label=alpha,color=red,width=4.5
marker = type=square,lat=42.358,lon=-71.0874,color=blue,width=8
```

Each entry is a string of comma-separated pairs. The order is not significant. The only mandatory fields are for the marker type and position. The position can be given in local x-y coordinates or in earth coordinates. If both are given for some reason, the earth coordinates will take precedent. The width parameter is given in meters drawn to scale on the geo display. Shapes are roughly 10x10

meters by default. The GUI provides a hook to scale all markers globally with the ALT-m and CTRL-m hot keys and in the GeoAttributes pull-down menu.

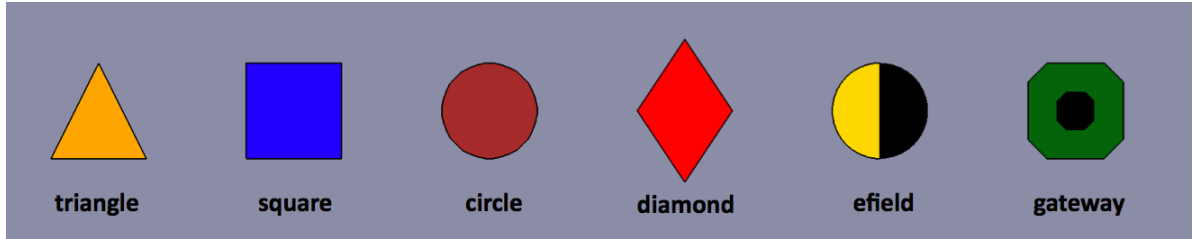


Figure 11: **Markers:** Types of markers known to the pMarineViewer.

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

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

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

The effect of a "moving" marker, or a marker that changes color, can be achieved by repeatedly publishing to the VIEW\_MARKER variable with only the position or color changing while leaving the label and type the same. To dynamically alter the text rendered with a marker, the msg=value field may be used instead. When the message is non-empty, it will be rendered instead of the label text.

### 5.3 Comms Pulses

Comms pulse objects were designed to convey a passing of information from one node to another. At this writing, they are only used by the uFldNodeComms application, but from the perspective of pMarineViewer it does not matter the origin. The MOOS variable is VIEW\_COMMS\_PULSE. They look something like that shown in Figure 12. There are two pulses shown in this figure. In this case they were posted by uFldNodeComms to indicate that the two vehicles are receiving each other's node reports.

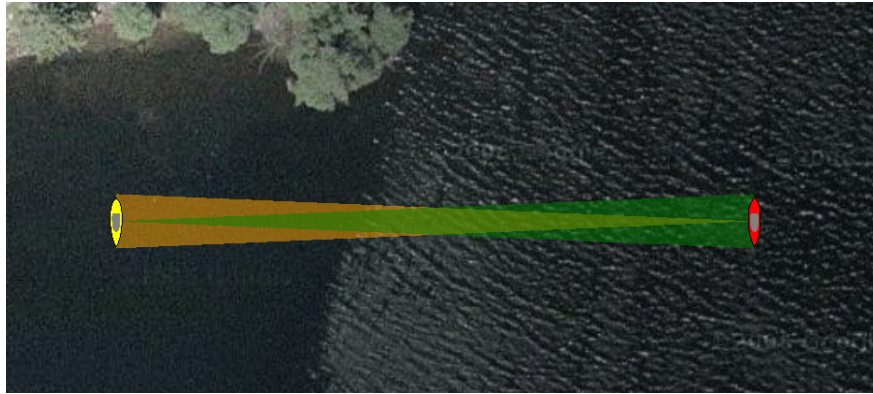


Figure 12: **Comms Pulses:** A comms pulse directionally renders communication between vehicles. Here each vehicle is communicating with the other, and two different colored pulses are rendered.

The term "pulse" is used because the object has a duration (by default three seconds), after which it will no longer be rendered by `pMarineViewer`. The pulse will fade (become more transparent) linearly with time as it approaches its expiration. If a subsequent comms pulse is received with an identical label before the first pulse times out, the second pulse will replace the first, in the style of other geometric objects discussed previously. Although serializing and de-serializing comms pulse messages is outside the scope of this discussion, it is worth examining an example comms pulse message:

```
VIEW_COMMS_PULSE = sx=91,sy=29,tx=6.7,ty=1.4,beam_width=7,duration=10,fill=0.35,
                    label=GILDA2HENRY_MSG,fill_color=white,time=1350201497.27
```

As with the object types discussed previously, the construction of the above type messages should be handled by the `XYCommsPulse` class along the line of something like:

```
#include "XYCommsPulse.h"

XYCommsPulse my_pulse(91, 29, 6.7, 1.4);
my_pulse.set_label("GILDA2HENRY_MSG");
my_pulse.set_duration(10);
my_pulse.set_beam_width(7);
my_pulse.set_fill(0.35);
my_pulse.set_color("fill", "white");
string str = my_pulse.get_spec();
m_Comms.Notify("VIEW_COMMS_PULSE", str);
```

The white comms pulse shown in Figure 13 indicates that a message has been sent from one vehicle to the other. The fat end of the pulse indicates the receiving vehicle. The color scheme is not a convention of `pMarineViewer`, but rather a convention of the `uFldNodeComms` application which generated the object in this case. A white pulse is typically rendered long enough to allow the user to visually register the information. It also typically does not move with the vehicle, to convey to the user the vehicle positions at the time of the communication.



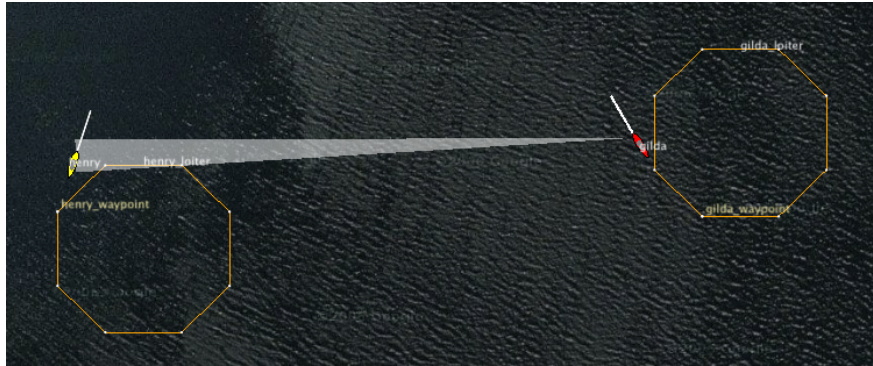


Figure 13: **Comms Pulses for Messaging:** In this figure the white comms pulse indicates that a message is being sent from one vehicle to another, via `uFldNodeComms`.

The rendering of comms pulses may be toggled on or off in `pMarineViewer` via a selection in the `GeoAttr` pull-down menu, or via the '@' hot key. It is not possible in `pMarineViewer` to show just the white comms pulses, and hide the colored node report comms pulses, or vice versa. It is possible however in the `uFldNodeComms` configuration to shut off the node report pulses with `view_node_rpt_pulses=false`.

## 5.4 Range Pulses

Range pulse objects were designed to convey a passing of information or sensor energy from one node to any other node in the vicinity, up to a certain range. At this writing they are only used by the `uFldContactRangeSensor` and `uFldBeaconRangeSensor` applications, but from the perspective of `pMarineViewer` it does not matter the origin. The MOOS variable is `VIEW_RANGE_PULSE`. They look something like that shown in Figure 14. Here the pulse is shown over three successive times.

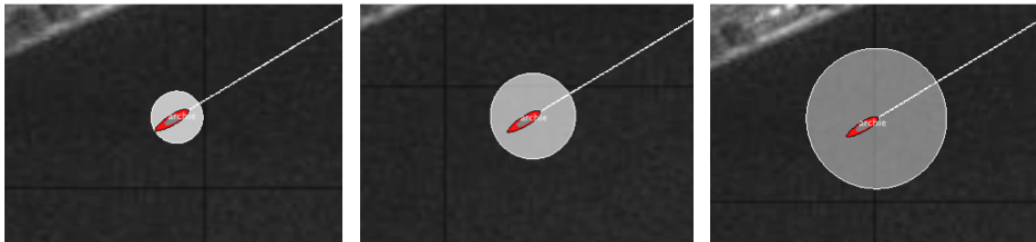


Figure 14: **Comms Pulses:** A comms pulse directionally renders communication between vehicles. Here each vehicle is communicating with the other, and two different colored pulses are rendered.

The term "pulse" is used because the object has a duration (by default 15 seconds), after which it will no longer be rendered by `pMarineViewer`. The pulse will grow in size and fade (become more transparent) linearly with time as it approaches its expiration. If a subsequent range pulse is received with an identical label before the first pulse times out, the second pulse will replace the first, in the style of other geometric objects discussed previously. Although serializing and de-serializing range pulse messages is outside the scope of this discussion, it is worth examining an example range pulse message:

```
VIEW_RANGE_PULSE = x=99.2,y=68.9,radius=50,duration=6,fill=0.9,label=archie_ping,  
edge_color=white,fill_color=white,time=2700438154.35,edge_size=1
```

As with the object types discussed previously, the construction of the above type messages should be handled by the `XYRangePulse` class along the line of something like:

```
#include "XYRangePulse.h"  
  
XYRangePulse my_pulse(99.2, 68.9);  
my_pulse.set_label("archie_ping");  
my_pulse.set_duration(6);  
my_pules.set_edge_size(1);  
my_pules.set_radius(50);  
my_pules.set_fill(0.9);  
my_pulse.set_color("edge", "white");  
my_pulse.set_color("fill", "white");  
string str = my_pulse.get_spec();  
m_Comms.Notify("VIEW_RANGE_PULSE", str);
```

## 5.5 Drop Points

A user may be interested in determining the coordinates of a point in the geo portion of the `pMarineViewer` window. The mouse may be moved over the window and when holding the `SHIFT` key, the point under the mouse will indicate the coordinates in the local grid. When holding the `CTRL` key, the point under the coordinates are shown in lat/lon coordinates. The coordinates are updated as the mouse moves and disappear thereafter or when the `SHIFT` or `CTRL` keys are release. Drop points may be left on the screen by hitting the left mouse button at any time. The point with coordinates will remain rendered until cleared or toggled off. Each click leaves a new point, as shown in Figure 15.

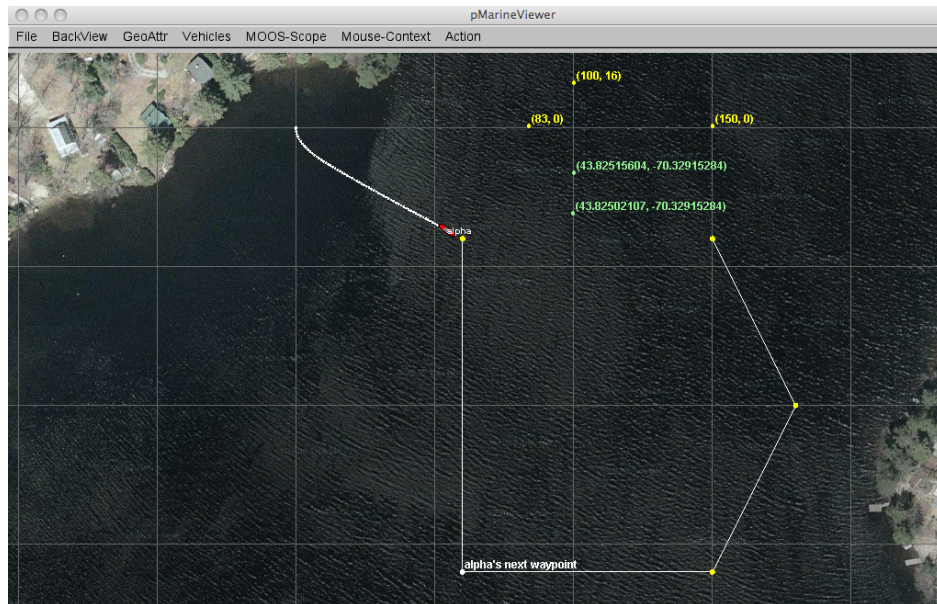


Figure 15: **Drop points:** A user may leave drop points with coordinates on the geo portion of the **pMarineViewer** window. The points may be rendered in local coordinates or in lat/lon coordinates. The points are added by clicking the left mouse button while holding the **SHIFT** key or **CTRL** key. The rendering of points may be toggled on/off, cleared in their entirety, or reduced by popping the last dropped point.

Parameters regarding drop points are accessible from the **GeoAttr** pull-down menu. The rendering of drop points may be toggled on/off by hitting the 'r' key. Drop points may also be shut off in the mission configuration file with `drop_point_viewable_all=false`. The set of drop points may be cleared in its entirety via the pull-down menu. Or the most recently dropped point may be removed by typing the **CTRL-r** key. The pull-down menu may also be used to change the rendering of coordinates from "as-dropped" where some points are in local coordinates and others in lat/lon coordinates, to "local-grid" where all coordinates are rendered in the local grid, or "lat-lon" where all coordinates are rendered in the lat/lon format. By default the mode is "as-dropped". The startup default mode may be changed with `drop_point_coords=local-grid` for example in the mission file.

## 6 The Vehicles Pull-Down Menu

The *Vehicles* pull-down menu deals with rendering properties of vehicles, vehicle labels, and vehicle trails. The options are shown in Figure 16. The very first option is to turn on or off the rendering of all vehicles. This can be done at run time via the menu selection, or toggled with the `Ctrl-’a’` hot key. Like all blue options in this menu, the text in the menu item may be placed verbatim in the mission configuration file to reflect the user’s startup preferences.

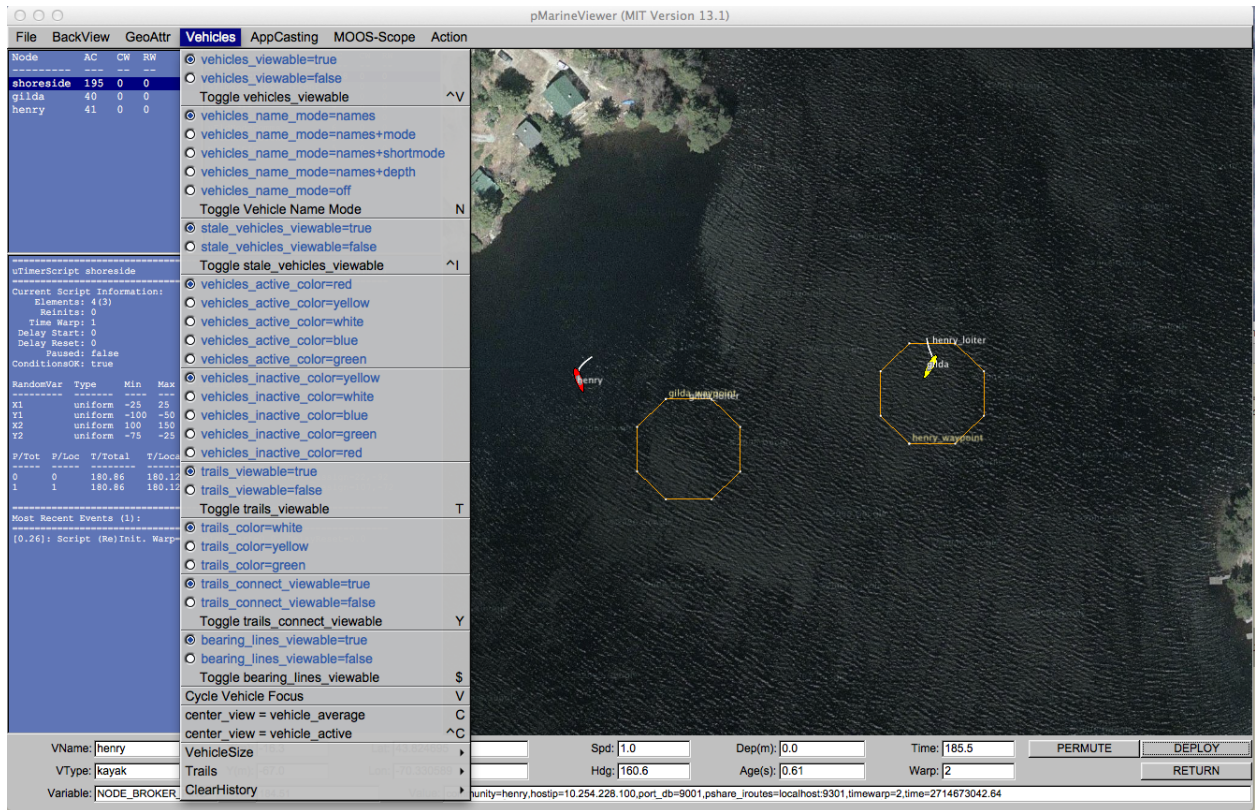


Figure 16: **The Vehicles menu:** This pull-down menu lists the options, with hot-keys, for affecting rendering vehicles and vehicle track history.

### 6.1 The Vehicle Name Mode

Each vehicle rendered in the viewer has an optional label rendered with it. This label may be rendered in one of five modes:

- *names*: Just the vehicle name is rendered.
- *names+mode*: The vehicle name and the full helm mode is rendered.
- *names+shortmode*: The vehicle name and the short helm mode is rendered.
- *names+depth*: The vehicle name and its current depth are rendered.
- *off*: No label is rendered.

The default is *names+shortmode*. The *names*, *off* and *depth* modes are self explanatory. The *names+mode* and *names+shortmode* involve information typically provided in vehicle node reports

about the state of the IvP helm. The helm uses hierarchical mode declarations as a way of configuring behaviors for missions. The helm mode for example be described with string looking something like "MODE@ACTIVE:LOITERING". In `pMarineViewer` the text next to the vehicle would be either this whole string if configured with the `names+mode` setting, or just "LOITERING" if configured with the `names+shortmode` setting.

The color of the rendered text may be changed from the default of white to any color in the Color Appendix with the `vehicles.name.color` configuration parameter.

## 6.2 Dealing with Stale Vehicles

A stale vehicle is one who has not been heard from for a some time, perhaps because the vehicle is disabled, out of range, or recovered from the field. These vehicles can be a distraction. Their history may be outright cleared as described in Section 6.6, but this requires action by the user or a posting to the MOOSDB.

Stale vehicles are also automatically dealt with by `pMarineViewer` in another way. After some number of seconds (30 by default), the vehicle label indicates the staleness. The label may look something like "henry (Stale Report: 231)" where the number indicates the number of seconds since the last node report received for this vehicle. After another period of time (30 by default), the vehicle may no longer be rendered and removed from the appcating pane.

A few features of this policy are configurable through the mission configuration file. The duration of time after which a vehicle is reported as stale may be changed from its default of 30 seconds with the `stale.report.thresh` parameter. The duration of time after which a vehicle is removed may be changed from its default of 30 seconds with the `stale.remove.thresh` parameter.

Stale vehicles are handled a bit differently when running in simulation and when running vehicles in the field. The difference between the two is determined by the MOOS time warp. Although it's possible to simulate with a time warp of one, here a time warp of one is interpreted as running physical vehicles. Simulated vehicles will be automatically removed from the viewer after `stale.report.thresh` + `stale.remove.thresh` seconds. When running actual vehicles, stale vehicles must be explicitly removed using the `alt+'k'` key to remove all stale vehicles, or `ctrl+'k'` key to remove the currently selected vehicle in the appcast pane.

## 6.3 Supported Vehicle Shapes

The shape rendered for a particular vehicle depends on the *type* of vehicle indicated in the node report received in `pMarineViewer`. There are several types that are currently handled:

- `kayak`
- `uuv`
- `auv` (same as `uuv`)
- `glider`
- `mokai`
- `ship`
- `longship`

- heron
- buoy
- swimmer

Note: the `swimmer`, `buoy`, `cray`, `bcray`, `crayx`, and `bcrays`, types were introduced *after* release 22.8.

Some shapes are shown in Figure 17.

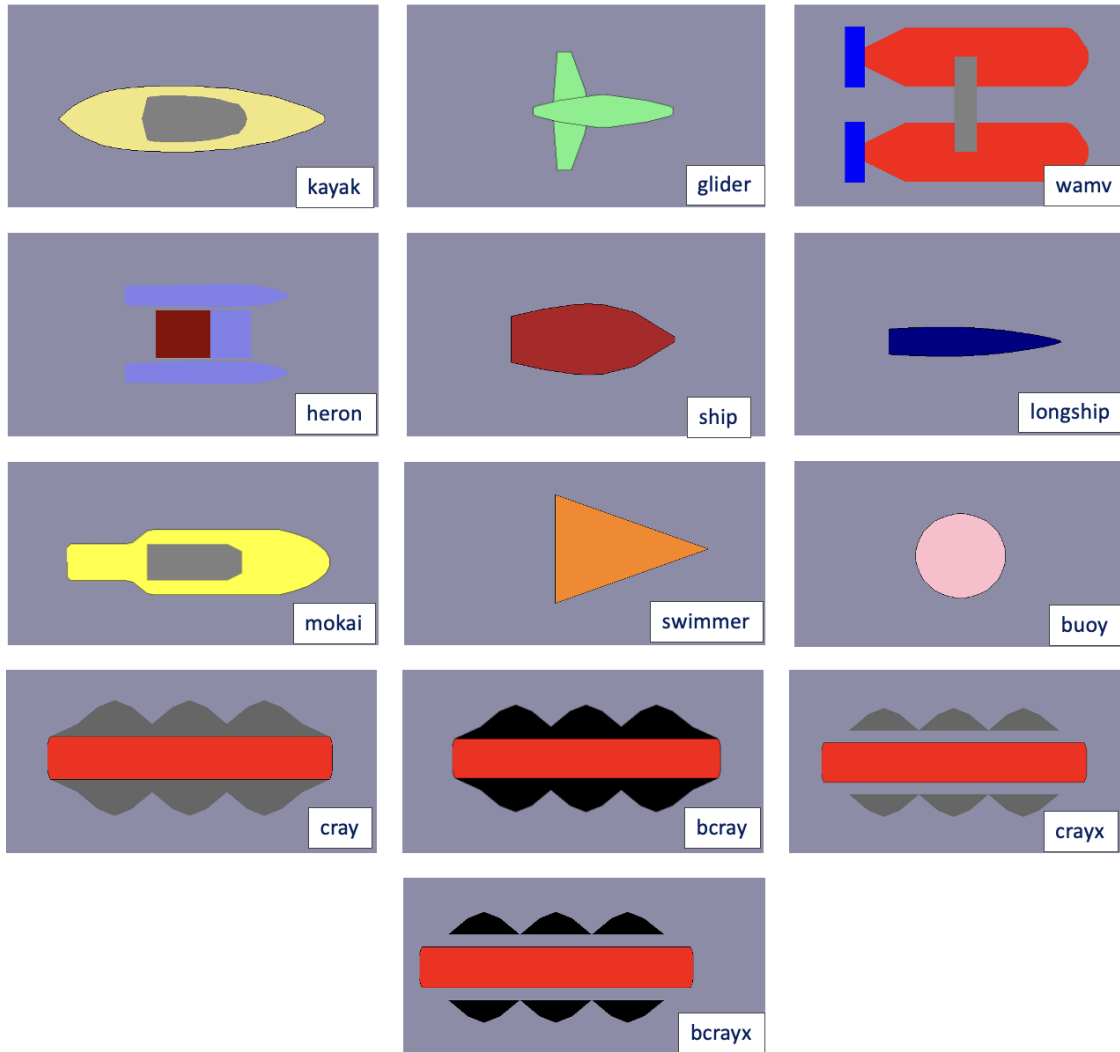


Figure 17: **Vehicles:** Types of vehicle shapes supported by `pMarineViewer`. The C-Ray shapes were added *after* release 22.8.

The default shape for an unknown vehicle type is currently set to be the shape "ship".

## 6.4 Vehicle Colors

Vehicles are rendered in one of two colors, the *active vehicle color* and the *inactive vehicle color*. The active vehicle is the one whose data is being rendered in the data fields at the bottom of the `pMarineViewer` window, and whose name is in the `VName:` field. The active vehicle may be changed by selecting "Cycle Vehicle Focus" from the Vehicles pull-down menu, or toggling through with the 'v' key. The default color for the active vehicle is red, and the default for the inactive vehicle is yellow. These can be changed via the pull-down menu, or with the following parameters in the configuration file:

```
vehicles_active_color   = <color>    // default is red
vehicles_inactive_color = <color>    // default is yellow
```

The parameters and colors are case insensitive. All colors of the form described in the Color Appendix are acceptable.

## 6.5 Centering the Image According to Vehicle Positions

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

## 6.6 Vehicle Trails

Vehicle trail (track history) rendering can be toggled off and on with the 't' key. The default is on. The startup default setting may be changed to off in the mission configuration file with `trails_viewable=false`.

### Trail Color and Point Size

The trail color by default is white. A few other colors are available in the Vehicles pull-down menu. A color may also be chosen in the mission configuration file with `trail_color=<color>` using any color listed in the Color Appendix. The trail point size may range from [1, 10]. The default setting is 2. The size may be changed at runtime by the Vehicles/Trails pull-down menu, or with the '{' and '}' hot keys. The startup trail size may also be set in the mission configuration file with `trails_point_size=<int>` parameter.

### Trail Length and Connectivity

Trails have a fixed-length history by default of 100 points. This may be changed via the Vehicles/Trails pull-down menu, or with the hot keys '(' and ')'. The startup default length may also be set in

the mission configuration file with `trails_length=<int>` with values in range of [0, 10000].

Individual trail points can be rendered with a line connecting each point, or by just showing the points. When the node report stream is flowing quickly, typically the user doesn't need or want to connect the points. When the viewer is accepting input from an AUV with perhaps a minute or longer delay in between reports, the connecting of points is helpful. This setting can be toggled with the 'y' or key, with the default being off. The startup default may be set to on with the mission file parameter `trails_connect_viewable=true`.

## Resetting or Clearing the Trails

A vehicle's history sometimes needs to be cleared, for example when a vehicle has not been heard from in a long time, or has been recovered. Its trails and other geometric objects posted to the viewer can become a distraction. This may be done in a couple ways. First via the Action pull-down menu, the last menu item allows the user to clear the history of all vehicles or a selected vehicle. The `Ctrl-9` hot key can be used to clear all vehicle histories. A select vehicle history may also be cleared by posting to the MOOS variable `TRAIL_RESET` with the name of the vehicle.

## 7 The InfoCasting Pull-Down Menu

Infocasting refers to two optional modes for operating `pMarineViewer`. The first is *appcasting*, introduced in 2012 in Release 12.11. The second mode is *realmcasting*, developed in December 2020 for the first release following 19.8. With these tools, `pMarineViewer` has been augmented to serve as an appcast and realmcast viewer. Other apps may be used for appcast viewing, such as `uMAC`, and `uMACView`. Realmcast viewing is also supported in `uMACView`, but not `uMAC`.

Appcasting requires that each app be "appcast enabled", and virtually all MOOS apps distributed with MOOS-IvP are appcast enabled. The motivation for appcasting and how to build appcast enabled MOOS applications are discussed elsewhere in the appcasting documentation. Realmcasting, on the other hand, requires nothing of the user other than running an app called `pRealm` in each MOOS community. In the simplest case, `pRealm` can be run with no configuration parameters, allowing it to be very easily added to legacy missions.

Quick points to note:

- The term *infocasting* refers generally to include either appcasting or realmcasting.
- By default, when `pMarineViewer` launches, Infocasting panes on the left are rendering appcast content, typically with a white-on-indigo color scheme.
- The content of the infocasting panes can be switched between appcasting and realmcasting by hitting the 'a' key.
- The infocasting panes can be toggled/hidden entirely by hitting the `ctrl-'a'` key.

### 7.1 Turning On and Off InfoCast Viewing

The InfoCasting pull-down menu, shown in Figure 18 allows adjustments to be made to the infocast rendering and content. The very first set of menu options allows the user to control whether the infocasting panes are shown or not (toggling with `ctrl-'a'`, followed by options for controlling the



content, appcasting or realmcasting. As with all `pMarineViewer` pull-down menu items, if they are rendered in blue, then the same text can be placed in the `pMarineViewer` configuration block to be applied automatically at startup.

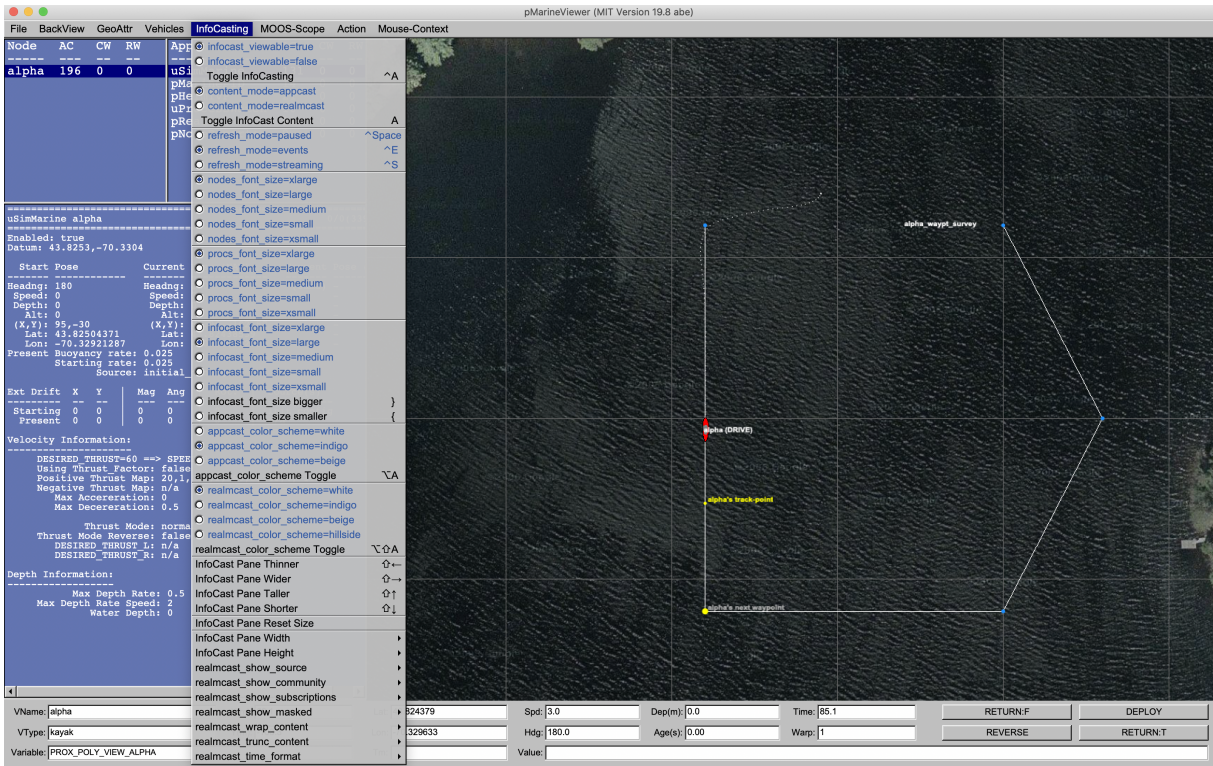


Figure 18: **The InfoCast menu:** This pull-down menu lists the options, with hot-keys, for affecting rendering aspects of the appcast panels, and policy for soliciting appcasts from known vehicles and applications.

## 7.2 Adjusting the InfoCast Viewing Panes Height and Width

The next set of menu items allow the relative size of the infocasting panes to be adjusted. The width of the three panes may be increased or decreased with the shift-left and shift-right arrow keys, and the height of the lower infocasting window relative to the two upper windows may be adjusted with the shift-up and shift-down arrow keys.

The infocast pane extents may also be set to the user's liking in the mission configuration file with the parameters `infocast_width` and `infocast_height`. The allowable range of values for each may be seen by pulling down the "InfoCast Pane Width" and "InfoCast Pane Height" sub-menus of the InfoCasting pull-down menu.

## 7.3 Adjusting the InfoCast Refresh Mode

The infocast *refresh mode* refers to the policy of sending appcast and realmcast requests to known vehicles and applications. This is discussed more fully in appcast documentation, but summarized here. Appcasting and Realmcasting are implemented to be lazy with respect to generating appcast

and infocast reports - they will not generate them unless asked. And even when asked, the request comes with an expiration after which, if no new request has been received, the application returns to the lazy mode of producing no appcasts. So, for `pMarineViewer` to function as an appcast and realmcast viewer, under the hood it must be also generating appcast requests (`APPCAST_REQ` postings) and realmcast requests (`REALMCAST_REQ` postings) to the `MOOSDB`. The refresh mode refers to this under-the-hood policy.

In the *paused* refresh mode, `pMarineViewer` is not generating any infocast requests at all. This is not the default and typically not very helpful, but it may be useful when the viewer is situated in the field with only a low-bandwidth connection to remote vehicles. The refresh mode may be set to *paused* via the pull-down menu selection, with the `CTRL+Spacebar` hot key, or set in the mission configuration file with `refresh_mode=paused`.

In the *events* refresh mode, the default mode, `pMarineViewer` is generating appcast requests only to the selected vehicle and the selected MOOS application. Even this is only partly true. In fact it is generating another kind of appcast request to all vehicles and apps, but this kind of request comes with the caveat that an app should only generate an appcast report if a new run warning has been detected. Otherwise these apps remain lazy. In this mode you should expect to see regular appcasts received for the selected app, and updates for the other apps only if something worthy of a run warning has occurred. You can confirm this for yourself by looking at the counter reflecting the number of appcasts received for any application. This counter is under the `AC` column in the upper panes. The refresh mode may be set to *events* via the pull-down menu selection, with the `CTRL+'e'` hot key, or set in the mission configuration file with `refresh_mode=events`. The latter would be redundant however since this is the default mode.

In the *streaming* refresh mode, `pMarineViewer` is generating appcast requests to all vehicles and all apps to generate appcasts all the time. This mode is a bandwidth hog, but it may be useful at times, especially to debug why a particular application is silent. If it is not generating an appcast in this mode, then something may indeed be wrong. The refresh mode may be set to *streaming* via the pull-down menu selection, with the `CTRL+'s'` hot key, or set in the mission configuration file with `refresh_mode=streaming`. The *streaming* mode is not relevant to realmcast generation. Realmcast generation may be only in either the paused for events mode.

## 7.4 Adjusting the InfoCast Fonts

The font size of the text in the infocasting panes may be adjusted. There are three panes:

- *Nodes Pane*: The upper left pane shows the list of nodes (typically synonymous with vehicles), presently known to the viewer.
- *Procs Pane*: The upper right pane show the list of apps, for the chosen node, presently known to the viewer.
- *InfoCast Pane*: The bottom pane shows the contents of the presently selected appcast or realmcast report.

For each pane the possible font settings are `xlarge`, `large`, `medium`, `small`, and `xsmall`. The default for the upper panes is `large`, and the default for the infocast pane is `medium`. Font sizes may

be changed via the pull-down menu or set to the user's liking in the mission configuration file with `nodes_font_size`, `procs_font_size`, and `infocast_font_size` parameters.

## 7.5 AppCasting Versus RealmCasting

Although both types of infocast content, appcast and realmcast reports, can be viewed from other client apps like `uMACView` and `uMAC`, they were originally and primarily designed to be accessed from within `pMarineViewer`. It is worth discussing here their similarities and differences. In distributed multi-vehicle simulations and in-field operations, the content available through infocasting is priceless for development and safe field operation. Much care was taken to allow the user access to an enormous variety of information across all vehicles, while ensuring that the information flow is restricted to be only what the user is presently monitoring through the current infocast pane. In both modes, if bandwidth is an issue, information flow can be halted immediately by hitting `ctrl-spacebar`.

With appcasting, the basic idea is shown in Figure 19. The `pMarineViewer` user selects a node (either the shoreside or one of the simulated or deployed vehicles), and a process/app running on that node. This information is contained in an `APPCAST_REQ` message published by `pMarineViewer` which is shared out to the particular node and app. If this app is appcast enabled (most are), it will respond with an `APPCAST` message. This single string message will be expanded to a multi-line report displayed in the infocast pane when received by `pMarineViewer`.

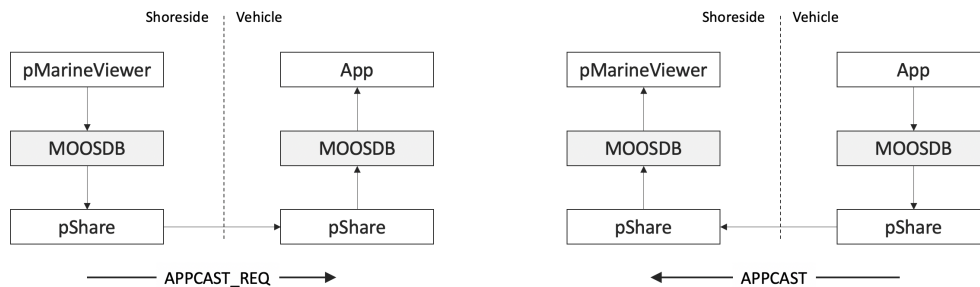


Figure 19: **Appcasting Information Flow:** An appcast viewing client, like `pMarineViewer`, `uMACView`, or `uMAC` will generate an appcast request to a targeted remote application. If the remote application is appcast enabled, it will respond with an appcast report for viewing in the appcast viewing client.

Realmcasting works a bit differently. It does not involve direct communication with remote apps as with appcasting. Instead, realmcasting is supported by a single instance of another app, `pRealm`. This app is run on each node, including the shoreside. This app is more fully described in [1]. It utilizes a special variable published by the local MOOSDB, `DB_RWSUMMARY` which informs `pRealm` what variables are subscribed for and published for each app connected to the MOOSDB. With this information, `pRealm` also subscribes for all variables and is able to form a realmcast report. Typically the form of this report is a summary, for a specified app, of all variables involved in subscriptions or publications. As with appcasting, the report generation is on-demand, requiring a realmcast request followed by a realmcast reply, as shown in Figure 20.

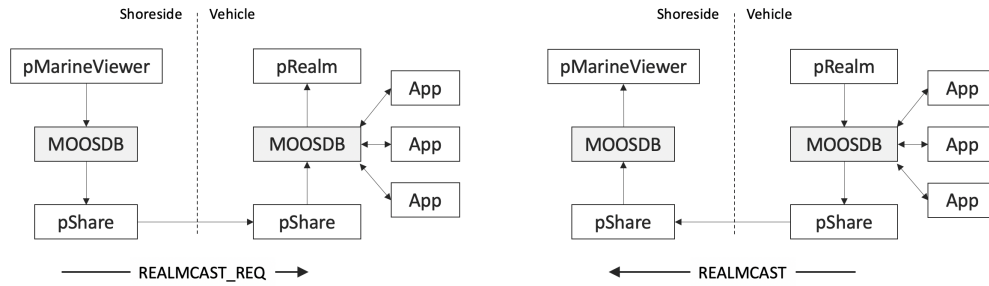


Figure 20: **Realmcasting Information Flow:** An realmcast viewing client, like `pMarineViewer` or `uMACView` will generate an realmcast request to a targeted node running an instance of `pRealm`. The realmcast request will specify a particular app and the realmcast response will report the status of variables involved in publications or subscriptions for that app.

The realmcast report is split into two parts. The top part shows all variables subscribed for by the given application, and the bottom part similarly shows all variables published by the application. In this way, a realmcast report is another way to glean the pub/sub interface for any application. A typical report is shown in Figure 21 below.

```

=====
pContactMgrV20          gilda (140)
=====
Subscriptions
=====
Variable      Source      Time      Comm      Value
-----
APPCAST_REQ   uMAC_220    241.52    shoreside node=all,app=all,duration=3.0,key=uMAC_220:app,thresh=run_warning
BCM_ALERT_REQUEST pHelmivP    0.28      gilda     id=avdcol_,on_flag=CONTACT_INFO=name=${VNAME} # contact=${VNAME},alert_range=80, cpa_range=85,match_type=mokai
NAV_HEADING    uSimMarine  242.21    gilda     180
NAV_SPEED      uSimMarine  242.21    gilda     0
NAV_X          uSimMarine  242.21    gilda     180
NAV_Y          uSimMarine  242.21    gilda     0
NODE_REPORT    uFldNodeComms 241.73    shoreside NAME=henry,X=0,Y=0,SPD=0,HDG=180,TYPE=kayak,MODE=PARK,ALLSTOP=ManualOverride,INDEX=479,TIME=3218502913.32,LENGTH=4
=====
Publications
=====
Variable      Source      Time      Comm      Value
-----
ALERT_VERBOSE  pContactMgrV20 2.91      gilda     new_contact=henry
APPCAST       uProcessWatch 7.68      gilda     formatted report
CONTACTS_LIST  pContactMgrV20 2.91      gilda     henry
CONTACTS_RECAP pContactMgrV20 242.05    gilda     vname=henry,range=180.00,age=0.69
CONTACT_CLOSEST pContactMgrV20 2.91      gilda     henry
CONTACT_CLOSEST_TIME pContactMgrV20 2.91      gilda     3218502674.874138
PCONTACTMGRV20_ITER_GAP pContactMgrV20 242.04    gilda     1.004112
PCONTACTMGRV20_ITER_LEN pContactMgrV20 242.05    gilda     0.000828
PCONTACTMGRV20_PID pContactMgrV20 -0.12     gilda     42130
PCONTACTMGRV20_STATUS pContactMgrV20 241.04    gilda     AppErrorFlag=false,Uptime=241.812,cpuload=0.2981,memory_kb=3224,memory_max_kb=3224,

```

Figure 21: **Example RealmCast Report:** A realmcast message is expanded into a multi-line report, consisting of a report on subscribed variables and published variables for a given application. In this case the report is for the `pContactMgrV20` application on vehicle `gilda`.

## 7.6 Adjusting the RealmCast Content

The content of the report can be adjusted by the client, e.g., `pMarineViewer` user, to help visualize the important information. There are seven options:

- **Source:** The Source column of the report may be suppressed.
- **Community:** The Community column of the report may be suppressed.
- **UTC Time:** The time format may be shown in absolute UTC time, or relative to the start of the local MOOSDB.

- Subscriptions: The subscriptions portion of the report may be suppressed.
- Mask: In the subscriptions portion, virgin variables may be suppressed.
- Wrap: Long string content may be wrapped over several lines, e.g., as in Figure 21.
- Truncate: Long string content may be truncated.

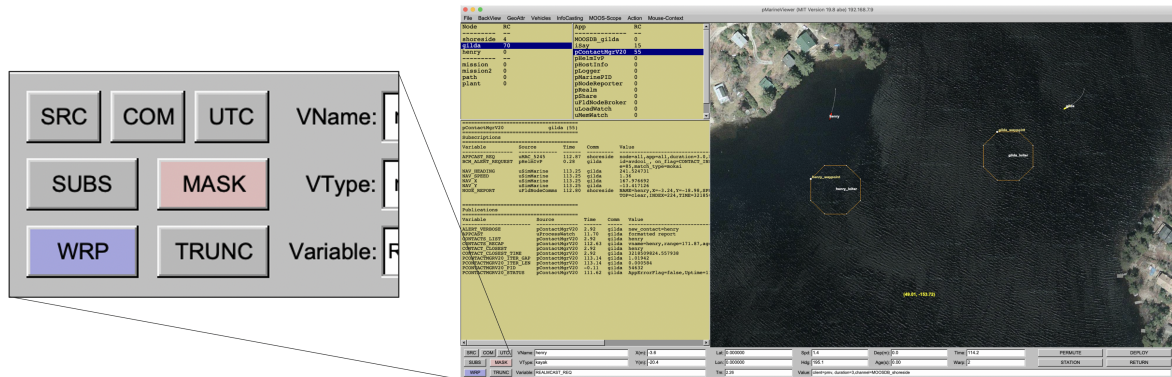


Figure 22: **Adjusting RealmCast Buttons:** Realmcast content may be adjusted through the seven buttons on the lower left corner of **pMarineViewer**. These buttons are only present when realmcasting is enabled.

The start-up value of these settings may also be set to the user's liking in the **pMarineViewer** configuration block:

```

realmcast_show_source      = true/false // Default is true
realmcast_show_community  = true/false // Default is true
realmcast_show_subscriptions = true/false // Default is true
realmcast_show_masked    = true/false // Default is true
realmcast_wrap_content    = true/false // Default is false
realmcast_trunc_content   = true/false // Default is false
realmcast_time_format_utc = true/false // Default is false

```

## 7.7 Additional RealmCast Capability: Watch Clusters

The normal realmcast reports allow the user to peer into any vehicle and any app to examine the state of variables involved in the subscriptions and publications for that app. However, in some multi-vehicle missions, it may be very useful to quickly see the value of certain variables for all vehicles *simultaneously*. For this purpose, **pMarineViewer** may be optionally configured with one or more *watch clusters*.

A watch cluster is a group of variables, with an associated grouping key, to allow simultaneous monitoring of these variable over all vehicles in the realmcast pane. This configuration is done on the client (**pMarineViewer**) side, and will result in a modified posting of **REALMCAST\_REQ**. The **pRealm** applications knows how to receive these requests, and will generate a posting to the **WATCHCAST** variable, one for each cluster variable, containing the most recent information for that variable.

We will explain by way of an example. Suppose we have a mission with six vehicles, `abe`, `ben`, `cal`, `deb`, `eve`, and `fin`. And we would like to monitor the local MOOS variables `DEPLOY`, `RETURN`, `STATION_KEEP`, and `COVER`. These four variables constitute our watch cluster, and we pick a keyname, say `mission_top`. The watch cluster would be configured with the following line in the `pMarineViewer` configuration block:

```
watch_cluster = key=mission_top, vars=DEPLOY:RETURN:STATION_KEEP:COVER
```

The watch cluster key, `mission_top`, should then appear in the Nodes pane when launched, as shown in Figure 23

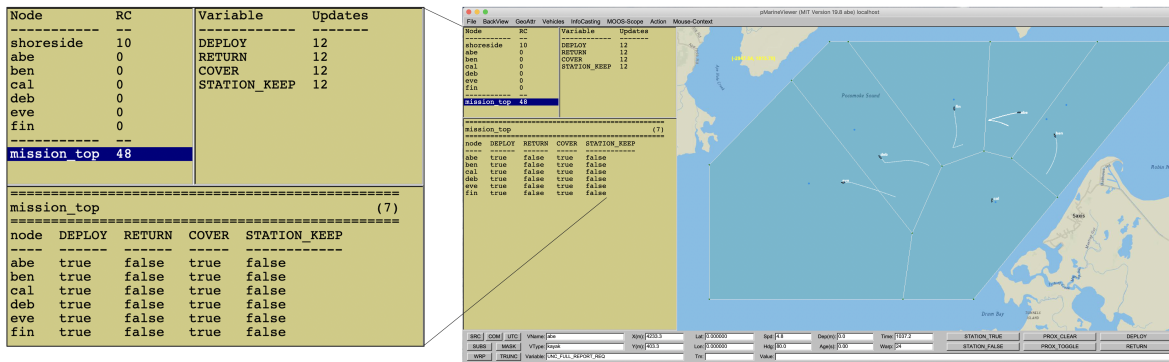


Figure 23: **RealmCast Watch Cluster Example:** A watch cluster has been configured with four variables, with the cluster name of `mission_top`. When the user selects `mission_top` in the nodes pane, the values of the four watched variables, for the six vehicles in the mission, are shown in a table in the bottom realmcast pane. This table is only refreshed when one or more of the variables has been changed on one or more of the vehicles.

The table in Figure 23 is quite concise, providing a substantial amount of information. However, it comes at the expense of leaving out certain information for each cell, such as the source, time, and community associated with the posting in that cell. For each variable in the cluster, the user has the option of finding out more. Notice the four cluster variables are listed in the upper right pane. By clicking on any one of them, the realmcast pane switches from the collective table format of Figure 23 to the single variable table, with examples shown in Figure 24 below.

| Node           | RC | Variable     | Updates |
|----------------|----|--------------|---------|
| shoreside      | 10 | DEPLOY       | 12      |
| abe            | 0  | RETURN       | 12      |
| ben            | 0  | COVER        | 12      |
| cal            | 0  | STATION_KEEP | 12      |
| deb            | 0  |              |         |
| eve            | 0  |              |         |
| fin            | 0  |              |         |
| mission_top 48 |    |              |         |

| Node | Var    | Time  | Source        | Comm      | Value |
|------|--------|-------|---------------|-----------|-------|
| abe  | DEPLOY | 59.31 | pMarineViewer | shoreside | true  |
| ben  | DEPLOY | 56.78 | pMarineViewer | shoreside | true  |
| cal  | DEPLOY | 53.87 | pMarineViewer | shoreside | true  |
| deb  | DEPLOY | 51.31 | pMarineViewer | shoreside | true  |
| eve  | DEPLOY | 48.77 | pMarineViewer | shoreside | true  |
| fin  | DEPLOY | 46.30 | pMarineViewer | shoreside | true  |

| Node           | RC | Variable     | Updates |
|----------------|----|--------------|---------|
| shoreside      | 10 | DEPLOY       | 12      |
| abe            | 0  | RETURN       | 12      |
| ben            | 0  | COVER        | 12      |
| cal            | 0  | STATION_KEEP | 12      |
| deb            | 0  |              |         |
| eve            | 0  |              |         |
| fin            | 0  |              |         |
| mission_top 48 |    |              |         |

| Node | Var    | Time  | Source        | Comm      | Value |
|------|--------|-------|---------------|-----------|-------|
| abe  | RETURN | 59.31 | pMarineViewer | shoreside | false |
| ben  | RETURN | 56.78 | pMarineViewer | shoreside | false |
| cal  | RETURN | 53.87 | pMarineViewer | shoreside | false |
| deb  | RETURN | 51.31 | pMarineViewer | shoreside | false |
| eve  | RETURN | 48.77 | pMarineViewer | shoreside | false |
| fin  | RETURN | 46.30 | pMarineViewer | shoreside | false |

| Node           | RC | Variable     | Updates |
|----------------|----|--------------|---------|
| shoreside      | 10 | DEPLOY       | 12      |
| abe            | 0  | RETURN       | 12      |
| ben            | 0  | COVER        | 12      |
| cal            | 0  | STATION_KEEP | 12      |
| deb            | 0  |              |         |
| eve            | 0  |              |         |
| fin            | 0  |              |         |
| mission_top 48 |    |              |         |

| Node | Var   | Time  | Source        | Comm      | Value |
|------|-------|-------|---------------|-----------|-------|
| abe  | COVER | 59.31 | pMarineViewer | shoreside | true  |
| ben  | COVER | 56.78 | pMarineViewer | shoreside | true  |
| cal  | COVER | 53.87 | pMarineViewer | shoreside | true  |
| deb  | COVER | 51.31 | pMarineViewer | shoreside | true  |
| eve  | COVER | 48.77 | pMarineViewer | shoreside | true  |
| fin  | COVER | 46.30 | pMarineViewer | shoreside | true  |

Figure 24: **Examining a Single Variable in a Watch Cluster:** When a single variable is selected in the watch cluster, the realmcast pane reports only on this variable, with one row for each vehicle. In this way, the user may see more information about the current posted value, such as the publisher (source), time posted, and the community of the app that posted the variable. When the user clicks again on the cluster keyname (in this case `mission_top`), the realmcast pane reverts to reporting on all variables in the cluster as in Figure 23.

## 7.8 Adjusting the AppCast and RealmCast Color Scheme

The infocast color schemes are by default different when the infocasting panes are in appcasting or realmcasting modes. There are a few different color schemes available to select from. The default appcast color scheme is "indigo", reflected for example in Figures 3 or 18. The default realmcast color scheme is "hillside", reflected for example in Figures 22 or 23. The color scheme may be changed by selecting a different option from the InfoCasting pull-down menu, or toggling through options with the hot key `Alt-'a'` for appcasting or `Alt-'A'` for realmcasting. Alternatively they may be configured in the `pMarineViewer` configuration block to have a different color scheme at launch time:

```
appcast_color_scheme = indigo/white/beige           // Default is indigo
realmcast_color_scheme = indigo/white/beige/hillside // Default is hillside
```

## 8 The MOOS-Scope Pull-Down Menu

The MOOS-Scope pull-down menu allows the user to configure `pMarineViewer` to scope on one or more variables in the `MOOSDB`. The viewer allows visual scoping on only a single variable at a time, but the user can select different variables via the pull-down menu, or toggle between the current and previous variable with the `'/'` key, or cycle between all registered variables with the `CTRL+'/'` key. The scope fields are on the bottom of the viewer as shown in Figure 25.

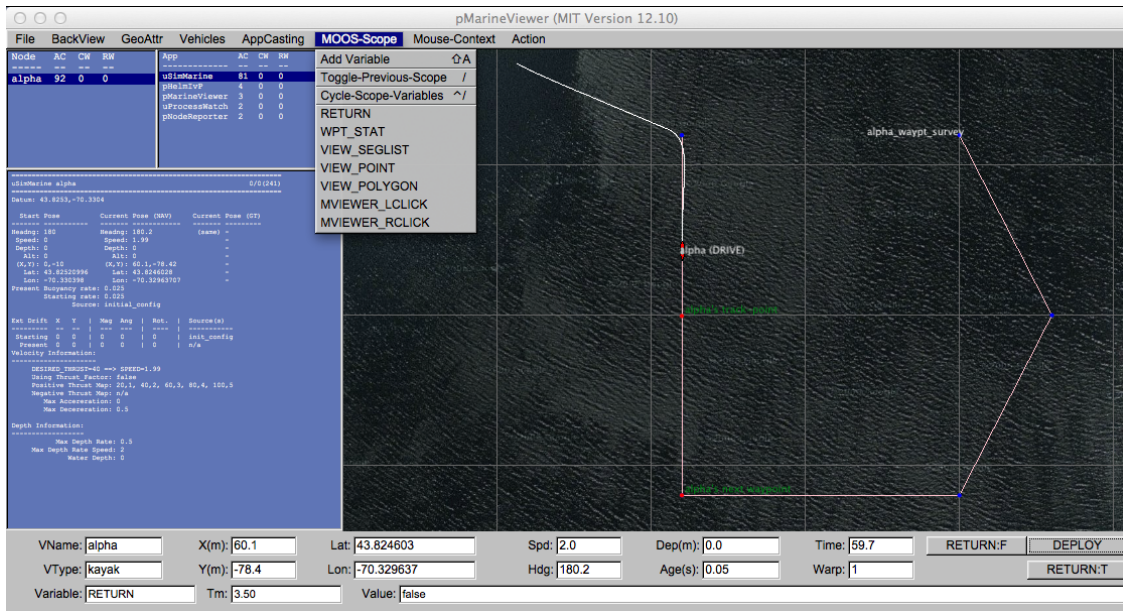


Figure 25: **The Scope Menu:** This pull-down menu allows the user to adjust which pre-configured MOOS variable is to be scoped, or to add a new variable to the scope list.

The three fields show (a) the variable name, (b) the last time it was updated, and (c) the current value of the variable. Configuration of the menu is done in the MOOS configuration block with entries similar to the following (which correlate to the particular items in the pull-down menu in Figure 25):

```
scope = RETURN, WPT_STAT, VIEW_SEGLIST, VIEW_POINT, VIEW_POLYGON
scope = MVIEWER_LCLICK, MVIEWER_RCLICK
```

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

## 9 The Exclusion Filter

A new feature added after Release 19.8.2: In situations where incoming information from a particular vehicle, or set of vehicles, is to be ignored, an *exclusion filter* may be used. `pMarineViewer` contains a filter that may be configured by specifying one or more vehicle names to ignore:

```
ignore_name = abe
ignore_name = abe, ben, cal
```

It may also be configured to consume information only by vehicles explicitly named:



```
match_name = ben, deb
match_name = abe
```

For an incoming posting to survive the exclusion filter it must (a) match any one of the list of match names, if any are provided, *and*, not match any one of the ignore names, if any are provided. Of course if no `match_name` or `ignore_name` parameters are provided, then nothing is filtered out.

The filter is applied to incoming node reports (`NODE_REPORT`), and incoming geometric messages such as `VIEW_POINT` or `VIEW_POLYGON`.

Note: Certain geometric objects such as `VIEW_COMMS_PULSE`, that are published by apps in the shoreside community, will not be filtered. Filtering is carried out by using the MOOS host community information of the incoming MOOS mail, which typically matches the vehicle name. Information such as `VIEW_COMSS_PULSE`, are generated by a shoreside MOOS app, such as `uFldNodeComms` in this case. As such, it will not be filtered.

## 10 The Action Pull-Down Menu

The Action pull-down menu allows the user to invoke pre-define pokes to the `MOOSDB` (the `MOOSDB` to which the `pMarineViewer` is connected). While hooks for a limited number of pokes are available by configuring on-screen buttons (Section 2.3), the number of buttons is limited to four. The "Action" pull-down menu allows for as many entries as will reasonably be shown on the screen. Each action, or poke, is given by a variable-value pair, and an optional grouping key. Configuration is done in the MOOS configuration block with entries of the following form:

```
action = menu_key=<key> # <MOOSVar>=<value> # <MOOSVar>=<value> # ...
```

If no such entries are provided, this pull-down menu will not appear. The fields to the right of the `action` are separated by the '#' character for convenience to allow several entries on one line. If one wants to use the '#' character in one of the variable values, putting double-quotes around the value will suffice to treat the '#' character as part of the value and not the separator. If the pair has the key word `menu_key` on the left, the value on the right is a key associated with all variable-value pairs on the line. When a menu selection is chosen that contains a key, then all variable-value pairs with that key are posted to the `MOOSDB`. The following configuration will result in the pull-down menu depicted in Figure 26.

```
action = menu_key=deploy # DEPLOY = true # RETURN = false
action+ = menu_key=deploy # MOOS_MANUAL_OVERRIDE=false
action = RETURN=true
```

The `action+` variant hints to the viewer that a line should be rendered in the pull-down menu separating it from following items.

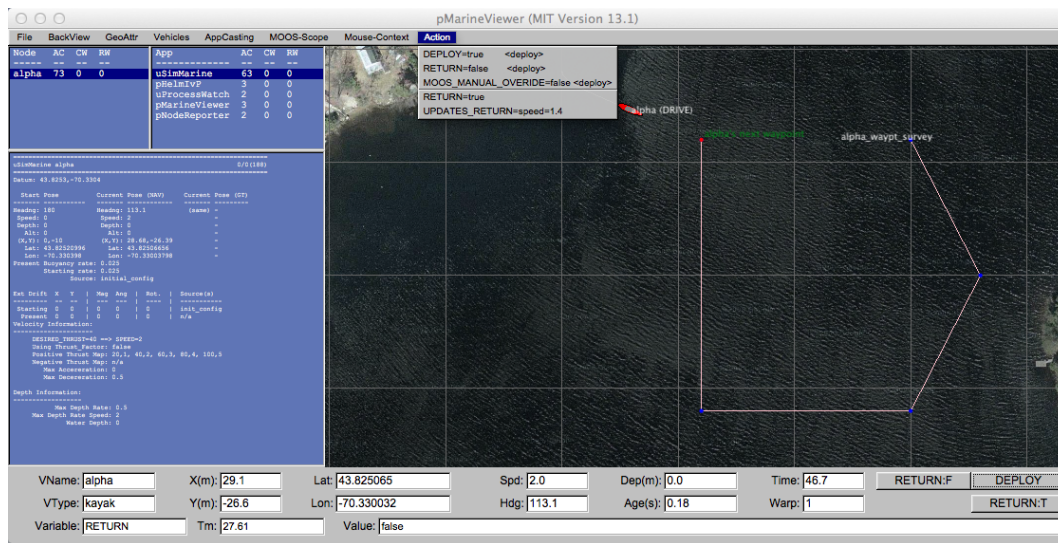


Figure 26: **The Action menu:** The variable value pairs on each menu item may be selected for poking or writing the MOOSDB. The three variable-value pairs above the menu divider will be poked in unison when any of the three are chosen, because they were configured with the same key, `<deploy>`, shown to the right on each item.

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

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

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

## 11 The Mouse-Context Pull-Down Menu

The Mouse-Context pull-down menu is an optional menu - it will not appear unless it is configured for use. It is used for changing the context of left and right mouse clicks on the operation area.

### 11.1 Generic Poking of the MOOSDB with the Operation Area Position

When the user clicks the left or right mouse in the geo portion of the pMarineViewer window, the variables `MVIEWER_LCLICK` and `MVIEWER_RCLICK` are published respectively with the operation area location of the mouse click, and the name of the active vehicle. A left mouse click may result in a publication similar to:

```
MVIEWER_LCLICK = x=19.0,y=57.0,lat=43.8248027,lon=-70.3290334,vname=henry,counter=1
```

A counter is maintained by `pMarineViewer` and is incremented and included on each post. The above style posting presents a generic way to convey to other MOOS applications an operation area position. In this case the other MOOS applications need to conform to this generic output. But, with a bit of further configuration, a similar *custom* post to the `MOOSDB` is possible to shift the burden of conformity away from the other MOOS applications where typically a user does not have the ability to change the interface.

## 11.2 Custom Poking of the MOOSDB with the Operation Area Position

Custom configuration of mouse clicks is possible by (a) allowing the MOOS variable and value to be defined by the user, and (b) exposing a few macros in the custom specification to embed operation area information. Configuration is done in the MOOS configuration block with entries of the following form:

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

The `left_context` and `right_context` keywords are case insensitive. If no entries are provided, this pull-down menu will not appear. The `<key>` component is optional and allows for groups of variable-data pairs with the same key to be posted together with the same mouse click. This is the selectable *context* in the Mouse-Context pull-down menu. If the `<key>` is empty, the defined posting will be made on all mouse clicks regardless of the grouping, as is the case with `MVIEWER.LCLICK` and `MVIEWER.RCLICK`.

Macros may be embedded in the string to allow the string to contain information on where the user clicked in the operation area. These patterns are: `$(XPOS)` and `$(YPOS)` for the local x and y position respectively, and `$(LAT)`, and `$(LON)` for the latitude and longitude positions. The pattern `$(IX)` will expand to an index (beginning with zero by default) that is incremented each time a click/poke is made. This index can be configured to start with any desired index with the `lclick_ix_start` and `rclick_ix_start` configuration parameters for the left and right mouse clicks respectively. The following configuration will result in the pull-down menu depicted in Figure 27.

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

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

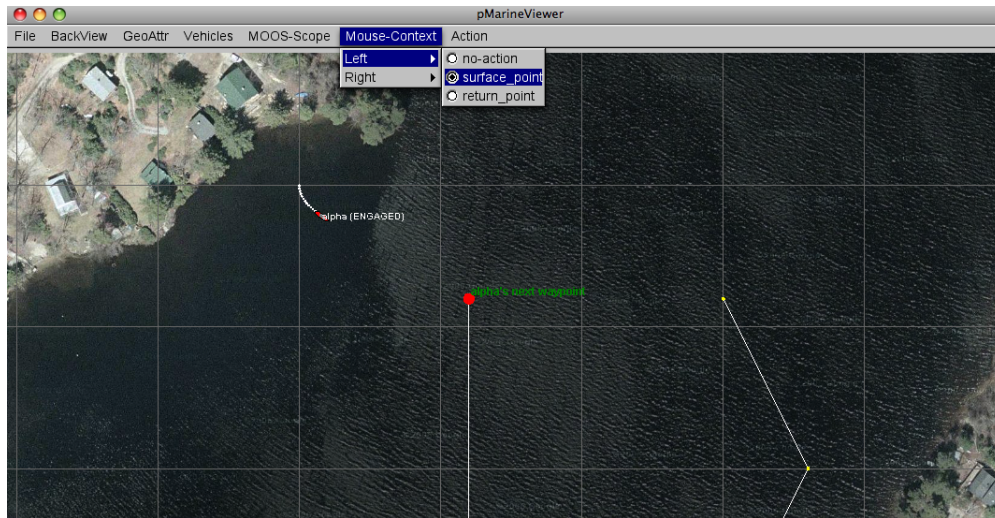


Figure 27: **The Mouse-Context menu:** Keywords selected from this menu will determine which groups of MOOS variables will be poked to the **MOOSDB** on left or mouse clicks. The variable values may have information embedded indicating the position of the mouse in the operation area at the time of the click.

The full set of macros are:

- `$(XPOS)`: The local x coordinate of the mouse click, to 0.1 meters.
- `$(X)`: The local x coordinate of the mouse click, to 1 meter.
- `$(YPOS)`: The local y coordinate of the mouse click, to 0.1 meters.
- `$(Y)`: The local y coordinate of the mouse click, to 1 meter.
- `$(IX)`: An integer that increments each each left mouse click. It starts at zero, but can be made to start at N with the config parameter `lclick.ix.start`.
- `$(BIX)`: An integer that increments each each right mouse click. It starts at zero, but can be made to start at N with the config parameter `rclick.ix.start`.
- `$(UTC)`: The current time in UTC seconds, in millisecond precision.
- `$(LAT)`: The local latitude coordinate of the mouse click.
- `$(LON)`: The local longitude coordinate of the mouse click.
- `$(VNAME)`: The active vehicle name
- `$(VNAME_CLOSEST)`: The name of the vehicle closest to the mouse click
- `$(UP_VNAME)`: The active vehicle name in upper case
- `$(UP_VNAME_CLOSEST)`: The name of the vehicle, in upper case, closest to the mouse click
- `$(HDG)`: The heading from the active vehicle to the x-y position of the mouse click.

Note macros above are of the form `$(MACRO)` using square brackets. Macros specified with parentheses, e.g., `$(MACRO)`, are also supported. The preferred form is the former, since parentheses macros are used by the `splug` utility for expanding mission files. Using square macros in `pMarineViewer` can help ward off confusion.

Certain macros are also available for expansion in the MOOS variable name component of a posting. For example, consider the following configuration entry:

```
left_context[return_point] = RETURN_POINT_${UP_VNAME_CLOSEST} = point=${XPOS},${YPOS}
```

the macro `_${UP_VNAME_CLOSEST}` will expand to the upper case name of the vehicle closest to the mouse click, and may result in a posting like:

```
RETURN_POINT_HENRY = point=87.2,99.8
```

The list of macros available for expansion in the MOOS variable name are:

- `_${VNAME}`: The active vehicle name
- `_${VNAME_CLOSEST}`: The name of the vehicle closest to the mouse click
- `_${UP_VNAME}`: The active vehicle name in upper case
- `_${UP_VNAME_CLOSEST}`: The name of the vehicle, in upper case, closest to the mouse click

## 12 Configuring and Using the Commander Pop-Up Window

The *commander pop-up* window is meant to provide an extendible command and control interface for missions requiring more than a few on-screen buttons. It needs to be configured and thought through prior to mission launch, to reflect whatever the mission operator may need to do during the mission. The window conveniently pops up and is hidden by toggling the space bar. An example is shown below.

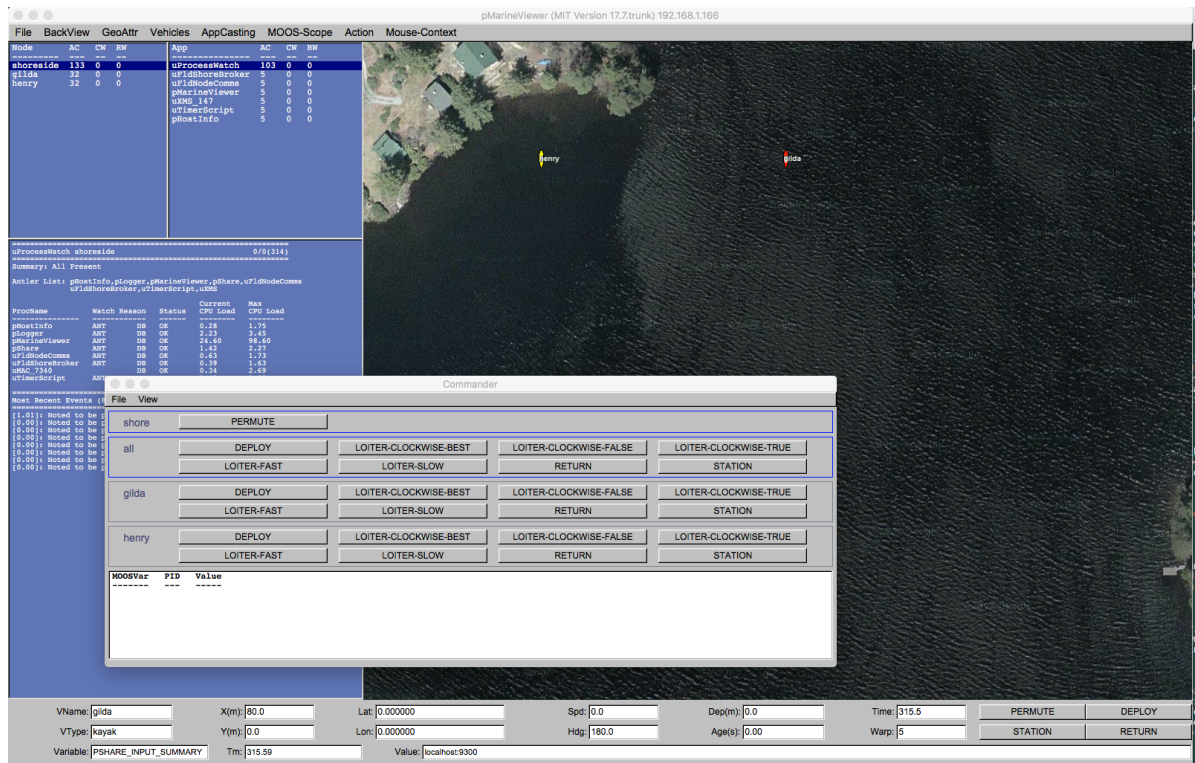


Figure 28: The commander pop-up window is toggled open/closed with the space-bar key and presents the user with a set of user-configured commands to either the all vehicles, individual vehicles, or the the pMarineViewer shoreside community itself.

Below we discuss how to work with the window and its content, how to configure the pop-up window, and the related assumptions of configuration in `uFldShoreBroker` to ensure proper command out to the vehicles.

## 12.1 Commander Pop-Up Window Actions and Content

The commander pop-up window has typically four components as indicated in Figure 29. The first component contains a single pane of buttons with postings to the local shoreside MOOSDB, not intended to be passed on to any vehicles. The second component contains a single pane of buttons for postings intended to be sent to all vehicles. The third component holds a dedicated pane for each known vehicle, with buttons intended for direct communication to the named vehicle. The fourth component contains the *command history*, a list of the actual postings made from recent button clicks, with the most recent postings at top.

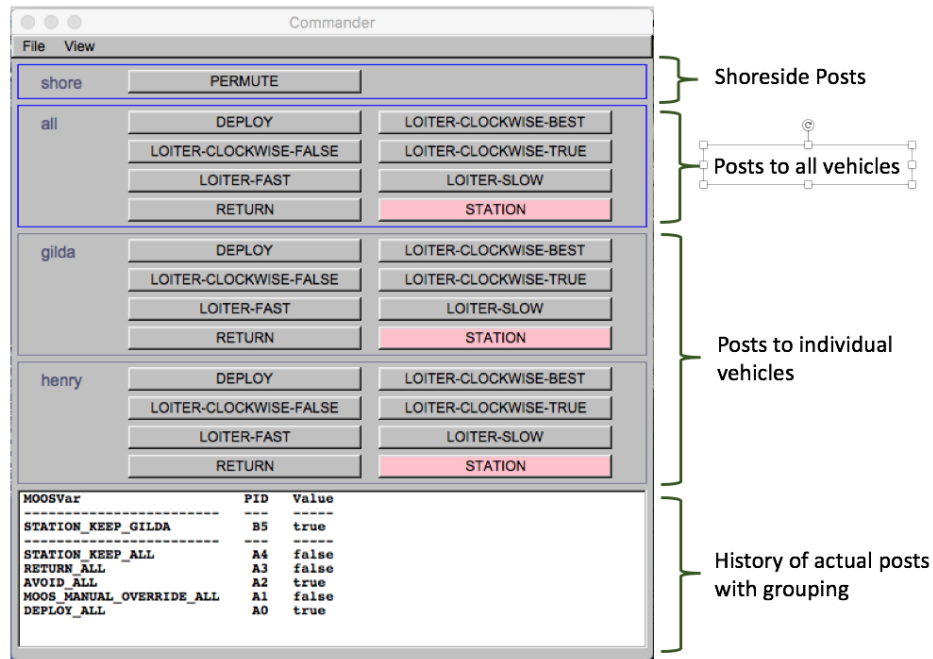


Figure 29: The commander pop-up window is contains four parts as shown. The button grouping is done automatically depending on the user configuration. Each button may be configured to make one or more posts to the MOOSDB.

The command history also indicates the clustering of posts associated with recent button clicks. The Post-ID, is shown in the middle column. All postings from the same button click, begin with the same letter, allowing the user to discern the group of postings resulting from a single button click. The user may also may a hypothetical post by clicking any button with the shift-key held down. In this case, no post will be made to the MOOSDB, but the actions that would have been posted are added to the command history, separated by a dashed line as in Figure 29. A subsequent actual button click would erase he hypothetical lines from the command history. Finally, if screen real estate is an issue, the command history pane may be hidden by toggling with the 'p' character.

The position and sizing of the command-popup window may be adjusted upon opening, with the

buttons retaining their size, but the button positions automatically adjusted to best fit the size of the window. As the user adjusts wider, the number of buttons in each row will grow, and the height will be as high as needed to fit the content. As the user adjusts taller, the command history pane is grown to show more history. The whole command pop-up window may be hidden and re-shown by toggling the spacebar-key. When doing so, note that not only the shape, but also the position w.r.t. the `pMarineViewer` window is also retained.

## 12.2 Commander Pop-Up Configuration

The set of possible command buttons and their corresponding posts to the MOOSDB may be referred to as a command portfolio, or *commandfolio* for short. This configuration is achieved with a set of lines of the following format:

```
cmd = label=<val>, var<val>, sval=<val>, dval=<val>, receivers=<val>, color=<val>
```

**The label field:** The button label is the text that will appear on the viewer button. It is also used as a unique identifier to associate several configuration lines together to make several posts with a single button.

**The var field:**

The `var` field names a single MOOS variable to be used in a post when this button is clicked. As such it should follow MOOS conventions and be upper case, no white space, not begin with a number, no use of special characters like '@', '?', '(' etc., and use the underscore character '\_' to separate components. These are only conventions and not enforced.

**The sval field:**

If a posting is to be made with a string value, the `sval` field should be used to specify this value.

**The dval field:** If a posting is to be made with a numerical (double) value, the `dval` field should be used to specify this value.

**The receivers field:** This field specifies a colon-separated list of vehicles, e.g., `henry:gilda` to be receivers of the posts associated with this button. If, for example, a vehicle named `henry` is named, with the MOOS variable `DEPLOY`, a button click will result in a posting to the variable `DEPLOY_HENRY`. Two special names are reserved, "all" and "shoreside". If "all" is used, `DEPLOY_ALL` will be posted, *not* `DEPLOY_GILDA` and `DEPLOY_HENRY` for example. If "shoreside" is used, simply `DEPLOY` will be posted.

**The colors field:**

If the user desires a certain command button to visually stand out, a color may be specified. For example, in Figure 29, the button to order a vehicle to station keep is highlighted in red, since this is often regarded as the go-to action when or if a surface vehicle encounters intervention from an operator.

### 12.3 Commander Pop-Up Example Configuration from m2\_berta Mission

Below is an example commander pop-up window configuration from the m2\_berta mission. It will result in the commander window shown in Figure 29. Note the macro \$(VNAMES) in many of the configuration lines. This macro is filled in at launch time by the colon-separated list of all vehicles being launched. See the launch script in the m2\_berta mission for an example.

```
cmd = label=DEPLOY, var=DEPLOY, sval=true, receivers=all:$(VNAMES)
cmd = label=DEPLOY, var=MOOS_MANUAL_OVERRIDE, sval=false, receivers=all:$(VNAMES)
cmd = label=DEPLOY, var=AVOID, sval=true, receivers=all:$(VNAMES)
cmd = label=DEPLOY, var=RETURN, sval=false, receivers=all:$(VNAMES)
cmd = label=DEPLOY, var=STATION_KEEP, sval=false, receivers=all:$(VNAMES)

cmd = label=RETURN, var=RETURN, sval=true, receivers=all:$(VNAMES)
cmd = label=RETURN, var=STATION_KEEP, sval=false, receivers=all:$(VNAMES)

cmd = label=PERMUTE, var=UTS_FORWARD, dval=0, receivers=shore

cmd = label=STATION, var=STATION_KEEP, sval=true, receivers=all:$(VNAMES), color=pink

cmd = label=LOITER-FAST, var=UP_LOITER, sval=speed=2.8, receivers=all:$(VNAMES)
cmd = label=LOITER-SLOW, var=UP_LOITER, sval=speed=1.4, receivers=all:$(VNAMES)

cmd = label=LOITER-CLOCKWISE-TRUE, var=UP_LOITER, sval=clockwise=true, receivers=all:$(VNAMES)
cmd = label=LOITER-CLOCKWISE-FALSE, var=UP_LOITER, sval=clockwise=false, receivers=all:$(VNAMES)
cmd = label=LOITER-CLOCKWISE-BEST, var=UP_LOITER, sval=clockwise=false, receivers=all:$(VNAMES)
```

### 12.4 Commander Pop-Up Coordination with pShare and uFldShoreBroker

Neither the commander popup window, nor `pMarineViewer` for that matter, are able to communicate with vehicles directly. The convention here is that if posting such as `DEPLOY=true` is desired to be sent to say vehicle "henry", then a post of `DEPLOY_HENRY=true` is made instead. And if all vehicles are to be deployed with a single button click, then a posting of `DEPLOY_ALL=true` is made.

The assumption is made that an inter-MOOSDB share arrangement of some form, typically with `pShare`, is made from the shoreside community to each vehicle. This can be made directly with `pShare` with a pair of lines as the following:

```
Output = src_name=DEPLOY_ALL, dest_name=DEPLOY, route=192.168.1.12:9300
Output = src_name=DEPLOY_GUS, dest_name=DEPLOY, route=192.168.1.12:9300
```

The above pair of lines would be needed for *each* vehicle command, for *each* vehicle. This is accomplished much more conveniently if the uField Toolbox tool `uFldShoreBroker` is used to automatically coordinate known vehicles, and commands with `pShare`. The above pair of lines is instead:

```
qbridge = DEPLOY
```

And for multiple commands, with say five vehicles for example:



```
qbridge = DEPLOY, RETURN, STATION_KEEP, UP_LOITER, MOOS_MANUAL_OVERRIDE
```

This single line using `uFldShoreBroker` would require 30 separate `pShare` lines.

## 13 Configuration Parameters for pMarineViewer

The blue items in pull-down menus are also available as mission file configuration parameters. The configuration parameter is identical to the pull-down menu text. For example in the BackView menu shown in Figure 7, the menu item `full_screen=true` may also be set in the `pMarineViewer` configuration block verbatim with `full_screen=true`.

### 13.1 Configuration Parameters for the BackView Menu

The parameters in Listing 2 relate to the BackView menu described more fully in Section 3. Parameters in blue below correlate to parameters in blue in the pull-down menu. For these parameters, the text in the pull-down menu is identical to a similar entry in the configuration file.

*Listing 13.2: Configuration Parameters for pMarineViewer BackView Menu.*

- `back_shade`: Shade of gray background when no image is used. Legal value range: [0, 1]. Zero is black, one is white.
- `full_screen`: If true, viewer is in full screen mode (no appcasts, no fields rendered at the bottom). Legal values: true, false. Section 3.4.
- `hash_delta`: Sets the hash line spacing. Legal values: 50, 100, 200, 500, 1000. The default is 100. Section 3.3.
- `hash_shade`: Shade of hash marks. Legal value range: [0, 1]. Zero is black, one is white. Section 3.3.
- `hash_viewable`: If true, hash lines are rendered over the op area. Legal values: true, false. The default is false. Section 3.3.
- `log_the_image`: If true, a request is posted to pLogger to log a copy of the image and info file. Legal values: true, false. The default is false. Section 3.2.
- `tiff_file`: Filename of a tiff file background image. After Release 22.8.x, multiple lines of this parameter will load multiple tiff files. Section 3.2.
- `tiff_file_b`: Filename of another tiff file background image. Deprecated after Release 22.8.x: just use multiple `tiff_file` lines. Section 3.2.
- `tiff_type`: Parameter no longer supported, after Release 22.8.x
- `tiff_viewable`: Use the tiff background image if set to true. Otherwise a gray screen is used as a background. Legal values: true, false. The default it true. Section 3.2.
- `view_center`: Sets the center of the viewing image in x,y local coordinates. Legal values: (double,double). The default is (0,0).

## 13.2 Configuration Parameters for the GeoAttributes Menu

The parameters in Listing 3 relate to the GeoAttributes pull-down menu described more fully in Section 5. Parameters in blue below correlate to parameters in blue in the pull-down menu. For these parameters, the text in the pull-down menu is identical to a similar entry in the configuration file.

*Listing 13.3: Configuration Parameters for pMarineViewer Geometry Menu.*

|  |   |
|--|---|
| <code>circle_viewable_all:</code>      | If false, circles are suppressed from rendering. Legal values: true, false. The default is true. Section 5.1.   |
| <code>circle_viewable_labels:</code>   | If false, circle labels are suppressed from rendering. Legal values: true, false. The default is true. Section 5.1.   |
| <code>comms_pulse_viewable_all:</code> | If false, comms pulses are suppressed from rendering. Legal values: true, false. The default is true. Section 5.3.  |
| <code>datum_viewable:</code>           | If false, the datum is suppressed from rendering. Legal values: true, false. The default is true. Sections 3.2 and 5.   |
| <code>datum_color:</code>              | The color used for rendering the datum. Legal values: Any color listed in the Colors Appendix. The default is red. Sections 3.2 and 5.  |
| <code>datum_size:</code>               | The size of the point used to render the datum. Legal values: Integers in the range [1, 10]. The default is 2. Sections 3.2 and 5.  |
| <code>drop_point_viewable_all:</code>  | If false, drop points are suppressed from rendering. Legal values: true, false. The default is true. Section 5.5.   |
| <code>drop_point_coords:</code>        | Specifies whether the drop point labels are in earth or local coordinates. Legal values are: as-dropped, lat-lon, local-grid. The default is as-dropped. Section 5.5.           |
| <code>drop_point_vertex_size:</code>   | The size of the point used to render a drop point. Legal values: Integers in the range [1, 10]. The default is 2. Section 5.5.  |
| <code>grid_viewable_all:</code>        | If false, grids are suppressed from rendering. Legal values: true, false. The default is true.  |
| <code>grid_viewable_labels:</code>     | If false, grid labels are suppressed from rendering. Legal values: true, false. The default is true.  |
| <code>grid_viewable_opaqueness:</code> | The degree to which grid renderings are opaque. Legal range: [0, 1]. The default is 0.3.  |
| <code>marker</code>                    | A marker may be stated in the configuration file with the same format of the <code>VIEW_MARKER</code> message. Section 5.2.   |
| <code>marker_scale:</code>             | The scale applied to marker renderings. Legal range: [0.1, 100]. The default is 1.0. Section 5.2.   |
| <code>marker_viewable_all:</code>      | If false, markers are suppressed from rendering. Legal values: true, false. The default is true. Section 5.2.   |
| <code>marker_edge_width:</code>        | Markers are rendered with an outer black edge. The edge may be set thicker to aid in viewing. Legal values: Integer values in the range [1, 10]. The default is 1. Section 5.2. |

|  |  |
|--|--|
| <code>marker_viewable_labels:</code>   | If false, marker labels are suppressed from rendering. Legal values: true, false. The default is true. Section 5.2.  |
| <code>oparea_viewable_all:</code>      | If false, oparea lines are suppressed from rendering. Legal values: true, false. The default is true.                |
| <code>oparea_viewable_labels:</code>   | If false, oparea label is suppressed from rendering. Legal values: true, false. The default is true.                 |
| <code>point_viewable_all:</code>       | If false, points are suppressed from rendering. Legal values: true, false. The default is true. Section 5.1.         |
| <code>point_viewable_labels:</code>    | If false, point labels are suppressed from rendering. Legal values: true, false. The default is true. Section 5.1.   |
| <code>polygon_viewable_all:</code>     | If false, polygons are suppressed from rendering. Legal values: true, false. The default is true. Section 5.1.       |
| <code>polygon_viewable_labels:</code>  | If false, polygon labels are suppressed from rendering. Legal values: true, false. The default is true. Section 5.1. |
| <code>range_pulse_viewable_all:</code> | If false, range pulses are suppressed from rendering. Legal values: true, false. The default is true. Section 5.3.   |
| <code>seglist_viewable_all:</code>     | If false, seglists are suppressed from rendering. Legal values: true, false. The default is true. Section 5.1.       |
| <code>seglist_viewable_labels:</code>  | If false, seglist labels are suppressed from rendering. Legal values: true, false. The default is true. Section 5.1. |
| <code>vector_viewable_all:</code>      | If false, vectors are suppressed from rendering. Legal values: true, false. The default is true. Section 5.1.        |
| <code>vector_viewable_labels:</code>   | If false, vector labels are suppressed from rendering. Legal values: true, false. The default is true. Section 5.1.  |

### 13.3 Configuration Parameters for the Vehicles Menu

The parameters in Listing 4 relate to the Vehicles pull-down menu described more fully in Section 6. Parameters in blue below correlate to parameters in blue in the pull-down menu. For these parameters, text in the pull-down menu is identical to a similar entry in the configuration file.

*Listing 13.4: Configuration Parameters for pMarineViewer Vehicles Pull-Down Menu.*

|                                      |  |
|--------------------------------------|--|
| <code>bearing_lines_viewable:</code> | If false, bearing lines will be suppressed from rendering. Legal values: true, false. The default is true.   |
| <code>center_view:</code>            | Sets the pan position to be either directly above the active vehicle, or the average of all vehicles. Legal values: active, average. The default is neither, resulting in the pan position being set to either (0,0) or set via other configuration parameters. Section 6.5. |
| <code>stale_remove_thresh:</code>    | Number of seconds after a stale vehicle has been detected before being removed. When time warp is one, vehicles are <i>not</i> automatically removed at all, and this number is meaningless. Legal values: Any non-negative number. The default is 30. Section 6.2.          |

|                                       |   |
|---------------------------------------|---|
| <code>stale_report_thresh:</code>     | Number of seconds after which a vehicle report will be considered stale. Legal values: Any non-negative number. The default is 60. Section 6.2.   |
| <code>trails_color:</code>            | The color of trail points rendered behind vehicles to indicate recent vehicle position history. Legal values: Any color listed in the Colors Appendix. The default is white. Section 6.6. |
| <code>trails_connect_viewable:</code> | If true the vehicle trail points are each connected by a line. Useful when node reports have large gaps in time. Legal values: true, false. The default is true. Section 6.6.             |
| <code>trails_length:</code>           | The number of points retained for the rendering of vehicle trails. Legal values: Integers in the range [1, 100000]. The default is 100. Section 6.6.                                      |
| <code>trails_point_size:</code>       | The size of the points rendering the vehicle trails. Legal values: Integers in the range [1, 10]. The default is 1. Section 6.6.  |
| <code>trails_viewable:</code>         | If false, vehicle trails are suppressed from rendering. Legal values: true, false. The default is true. Section 6.6.  |
| <code>vehicles_active_color:</code>   | The color of the active vehicle (the one who's data is being shown in the bottom data fields). Legal values: Any color listed in the Colors Appendix. The default is red. Section 6.4.    |
| <code>vehicles_inactive_color:</code> | The color of inactive vehicles. Legal values: Any color listed in the Colors Appendix. The default is yellow. Section 6.4.  |
| <code>vehicles_shape_scale:</code>    | The scale factor applied to vehicle size rendering. Legal values in the range: [0.1, 100]. The default is 1.0. Section 6.3.   |
| <code>vehicles_name_mode:</code>      | Sets the mode for rendering the vehicle label. Legal values are: names, names+mode, names+shortmode, names+depth, off. The default is names+shortmode. Section 6.1.                       |
| <code>vehicles_name_color:</code>     | Sets the color for rendering the vehicle label. Legal values are any color in the Colors Appendix. The default is white. Section 6.4.   |
| <code>vehicles_viewable:</code>       | If false, vehicles are suppressed from rendering. Legal values: true, false. The default is true. Section 6.  |

### 13.4 Configuration Parameters for the InfoCasting Menu

The parameters in Listing 5 relate to the InfoCasting pull-down menu described more fully in Section 7. Parameters in blue below correlate to parameters in blue in the pull-down menu. For these parameters, text in the pull-down menu is identical to a similar entry in the configuration file.

Note: Starting with the first release after 2019's Release 19.8, this `pMarineViewer` was augmented to include realmcasting in addition to appcasting. The term *infocasting* is the general term referring to either appcasting or realmcasting. As such, some of the earlier configuration parameters, e.g., `appcast_font_size`, have been replaced with a term like `infocast_font_size`. The older configuration parameters are deprecated, still supported for foreseeable releases.

*Listing 13.5: Configuration Parameters for pMarineViewer AppCast Pull-Down Menu.*

|  |   |
|--|---|
| <code>appcast_color_scheme:</code>         | The color scheme used in all three appcasting panes, affecting background color and font color. Possible settings, <code>default</code> , <code>beige</code> , <code>indigo</code> or <code>white</code> . The default is <code>indigo</code> . Section 7.8.    |
| <code>content_mode:</code>                 | Sets the on startup content, which can be changed by the user at any time. Either <code>appcast</code> or <code>realmcast</code> . The default is <code>appcast</code> .  |
| <code>infocast_font_size:</code>           | The font size uses in the <i>infocast</i> pane of the set of infocasting panes. Legal values: <code>xlarge</code> , <code>large</code> , <code>medium</code> , <code>small</code> , <code>xsmall</code> . The default is <code>medium</code> . Section 7.4.     |
| <code>infocast_height:</code>              | The height of the appcasting bottom pane as a percentage of the total <code>pMarineViewer</code> window height. Legal values: [30, 35, 40, 45,..., 85, 90]. The default is 75. Section 7.2.   |
| <code>infocast_viewable:</code>            | If <code>true</code> , the infocasting set of panes are rendered on the left side of the viewer. Legal values: <code>true</code> or <code>false</code> . The default is <code>true</code> . Section 7.1.  |
| <code>infocast_width:</code>               | The width of the infocasting panes as a percentage of the total <code>pMarineViewer</code> window width. Legal values: [20, 25, 30, 35,..., 65, 70]. The default is 30. Section 7.2.  |
| <code>nodes_font_size:</code>              | The font size used in the <i>nodes</i> pane of the set of infocasting panes. Possible settings, <code>xsmall</code> , <code>small</code> , <code>medium</code> , <code>large</code> , or <code>xlarge</code> . The default is <code>large</code> . Section 7.4. |
| <code>procs_font_size:</code>              | The font size used in the <i>procs</i> pane of the set of infocasting panes. Possible settings, <code>xsmall</code> , <code>small</code> , <code>medium</code> , <code>large</code> or <code>xlarge</code> . The default is <code>large</code> . Section 7.4.   |
| <code>realmcast_color_scheme:</code>       | Either <code>indigo</code> , <code>beige</code> , <code>hillside</code> , <code>white</code> , or <code>default</code> . The default is <code>hillside</code> . Section 7.8.  |
| <code>realmcast_show_source:</code>        | If <code>true</code> , the Source column is shown on realmcast output. Setting to <code>false</code> conserves screen space, possibly enhancing readability. The default is <code>true</code> . Section 7.6.  |
| <code>realmcast_show_community:</code>     | If <code>true</code> , the Community column is shown on realmcast output. Setting to <code>false</code> conserves screen space, possibly enhancing readability. The default is <code>true</code> . Section 7.6.   |
| <code>realmcast_show_subscriptions:</code> | If <code>true</code> , the Subscriptions block is shown on realmcast output. Setting to <code>false</code> conserves screen space, possibly enhancing readability. The default is <code>true</code> . Section 7.6.  |
| <code>realmcast_show_masked:</code>        | If <code>true</code> , certain variables are excluded in realmcast output, e.g., virgin variables. Setting to <code>false</code> conserves screen space, possibly enhancing readability. The default is <code>true</code> . Section 7.6.                        |
| <code>realmcast_wrap_content:</code>       | If <code>true</code> , the variable Value column in realmcast output will wrap onto several lines. Setting to <code>true</code> possibly enhances readability for very long output. The default is <code>false</code> . Section 7.6.                            |

- `realmcast_trunc_content`: If `true`, the variable Value column in realmcast output will be truncated. Setting to `true` possibly enhances readability for very long output. The default is `false`. Section 7.6.
- `realmcast_time_format_utc`: If `true` the Time column in realmcast output will be shown in UTC time instead of local time since app startup. The default is `false`. Section 7.6.
- `refresh_mode`: Determines the manner in which appcast requests are sent to apps. Legal values: `paused`, `events`, `streaming`. The default is `events`. Section 7.3.

## 13.5 Configuration Parameters for the Scope, MouseContext and Action Menus

*Listing 13.6: Configuration Parameters the Scope, MouseContext and Action Menus.*

- `scope`: A comma separated list of MOOS variables to scope. Section 8.
- `oparea`: A specification of the operation area boundary for optionally rendering.
- `button_one`: A configurable command and control button. The parameter `button_1` may also be used. Section 2.3.
- `button_two`: A configurable command and control button. The parameter `button_2` may also be used. Section 2.3.
- `button_three`: A configurable command and control button. The parameter `button_3` may also be used. Section 2.3.
- `button_four`: A configurable command and control button. The parameter `button_4` may also be used. Section 2.3.
- `button_five-twenty`: Additional configurable command and control button release following Release 19.8.1. Section 2.3.
- `action`: A MOOS variable-value pair for posting, available under the Action pull-down menu. Section 10.
- `left_context`: Allows the custom configuration of left mouse click context. Section 11.
- `right_context`: Allows the custom configuration of right mouse click context. Section 11.
- `lclick_ix_start`: Starting index for the left mouse index macro. Section 11.
- `rclick_ix_start`: Starting index for the right mouse index macro. Section 11.

## 13.6 Configuration Parameters for Optimizing in Extreme Load Situations

In missions with a very high number of vehicles and very high time warp, and perhaps also a very high number of geometric objects to render, `pMarineViewer` will strain, slow or even freeze. The breaking point is partly dependent on the speed and number of CPU cores on your machine. There are a few measures that can be taken to reduce the CPU load and squeeze more performance from `pMarineViewer`.

*Listing 13.7: Configuration Parameter for Optimizing with High Number of Vehicles.*

`node_report_unc`: If true, node reports will be ingested from `uFldNodeComms` rather than directly from the vehicles. (Introduced *after* Release 19.8.1.)

By setting the `node_report_unc` parameter to true, `pMarineViewer` will register for `NODE_REPORT_UNC` as its source for receiving node reports. Normally this information comes from `NODE_REPORT`. Normally the latter is shared directly from vehicles. A typical vehicle will produce this message at 4Hz. In simulations with a time warp of say 50, and simulating say 50 vehicles, this results in 10,000 node reports per second arriving in the mailbox of `pMarineViewer`. While this may be manageable, depending on the computer, it consumes a sizeable portion of the CPU load. If there are also happen to be very many geometric objects to render, then the workload will strain.

By ingesting node reports instead from `uFldNodeComms`, the rate of incoming node reports can be reduced to a level suitable for smooth trajectory rendering. In practice this seems to be about 10-20Hz before the human eye regards the motion as "jumpy". For the extreme case described above, the `uFldNodeComms` app can reduce the rate from 200Hz to 10Hz. Since `uFldNodeComms` is handling all these node reports anyway, and since it is likely relegated to a CPU different from `pMarineViewer` on multi-core machines, it is an effective way to aid `pMarineViewer`.

## 14 Publications and Subscriptions for pMarineViewer

The interface for `pMarineViewer`, in terms of publications and subscriptions, is described below. This same information may also be obtained from the terminal with:

```
$ pMarineViewer --interface or -i
```

### 14.1 Variables Published by pMarineViewer

It is possible to configure `pMarineViewer` to poke the `MOOSDB` via either the Action pull-down menu (Section 10), or via configurable GUI buttons (Section 2.3). It may also publish to the `MOOSDB` variables configured to mouse clicks (Section 11). So the list of variables that `pMarineViewer` publishes is somewhat user dependent, but the following few variables may be published in all configurations.

- `APPCAST`: Contains an appcast report identical to the terminal output. Appcasts are posted only after an appcast request is received from an appcast viewing utility.
- `APPCAST_REQ_<COMMUNITY>`: As an appcast viewer, `pMarineViewer` also generates outgoing appcast requests to MOOS communities it is aware of, including its own MOOS community. These postings are typically bridged to the other named MOOS community with the variable renamed simply to `APPCAST_REQ` when it arrives in the other community.
- `HELM_MAP_CLEAR`: This variable is published once when the viewer connects to the `MOOSDB`. It is used in the `pHelmIvP` application to clear a local buffer used to prevent successive identical publications to its variables.
- `PMV_CONNECT`: This variable is published once when the viewer connects to the `MOOSDB`. It is always published with the value 0. It can be used by any application that would like to publish or republish visual artifacts ensuring that `pMarineViewer` receives all mail in a sequence. Otherwise if the viewer comes up after the sequences is published, it may only get the last element in the sequence.

- **MVIEWER\_LCLICK**: When the user clicks the left mouse button, the position in local coordinates, along with the name of the active vehicle is reported. This can be used as a command and control hook as described in Section 11.
- **MVIEWER\_RCLICK**: This variable is published when the user clicks with the right mouse button. The same information is published as with the left click.
- **MVIEWER\_UNHANDLED\_MAIL**: If the viewer receives mail that it is unable to parse, info on the MOOS variable and its content will be posted in this variable for debugging.
- **PLOGGER\_CMD**: This variable is published with a "COPY\_FILE\_REQUEST" to log a copy of the image and info file, only if `log.the.image` is set to true. Section 3.2.

## 14.2 Variables Subscribed for by pMarineViewer

- **APPCAST**: As an appcast enabled *viewer*, **pMarineViewer** also subscribes for appcasts from other applications and communities to provide the content for its own viewing capability.
- **APPCAST\_REQ**: As an appcast enabled MOOS application, **pMarineViewer** also subscribes for appcast requests. Each incoming message is a request to generate and post a new appcast report, with reporting criteria, and expiration.
- **REALMCAST**: As an realmcast enabled *viewer*, **pMarineViewer** also subscribes for realmcasts from MOOS communities running **pRealm**, and renders them in the infocast panes described in Section 7.5.
- **WATCHCAST**: As an realmcast enabled *viewer*, **pMarineViewer** also subscribes for watchcasts from MOOS communities running **pRealm**, and renders them in the infocast panes described in Section 7.7.
- **NODE\_REPORT**: This is the primary variable consumed by **pMarineViewer** for collecting vehicle position information.
- **NODE\_REPORT\_LOCAL**: This serves the same purpose as the above variable. In some simulation cases this variable is used.
- **PHI\_HOST\_INFO**: A string representing the detected IP address, published by **pHostInfo**. Used for augmenting the **pMarineViewer** title bar with the current IP address.
- **TRAIL\_RESET**: When the viewer receives this variable it will clear the history of trail points associated with each vehicle. This is used when the viewer is run with a simulator and the vehicle position is reset and the trails become discontinuous.
- **VIEW\_CIRCLE**: A string representation of an XYCircle object.
- **VIEW\_COMMS\_PULSE**: A string representation of an XYCommsPulse object.
- **VIEW\_GRID**: A string representation of a XYConvexGrids object.
- **VIEW\_GRID\_CONFIG**: A string representation of a XYGrid configuration.
- **VIEW\_GRID\_DELTA**: A string representation of a XYGrid configuration.
- **VIEW\_POINT**: A string representation of an XYPoint object.
- **VIEW\_POLYGON**: A string representation of an XYPolygon object.
- **VIEW\_SEGLIST**: A string representation of an XYSegList object.
- **VIEW\_MARKER**: A string designation of a marker type, size and location.
- **VIEW\_RANGE\_PULSE**: A string representation of an XYRangePulse object.



- `VIEW_VECTOR`: A string representation of an `XYVector` object

## References

- [1] Michael R. Benjamin. pRealm: Integrated Scoping of the MOOSDB. <http://oceanai.mit.edu/ivpman/apps/pRealm>.