MIT 2.680

UNMANNED MARINE VEHICLE AUTONOMY, SENSING, AND COMMUNICATIONS

Lecture 4: MOOS Programming - Part II

Feb 12th, 2026

Web: http://oceanai.mit.edu/2.680

Email:
Mike Benjamin, mikerb@mit.edu

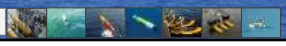


Photo by Arjan Vermeij, CMRE

1

Intro to MOOS Programming Outline

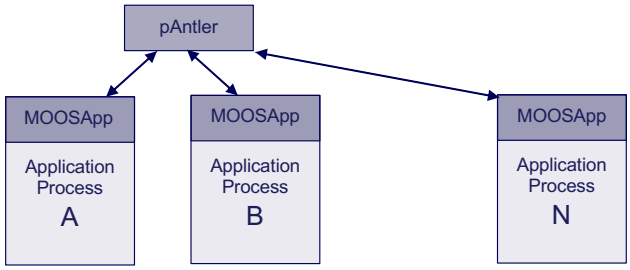
- Killing the MOOS
- Logging and Debugging
- MOOS and IvP C++ Utilities
- Common MOOS App Programming Practices

2

Killing the MOOS

- "Killing the MOOS" means ensuring that all MOOS processes have been killed
- Most missions involve the invocation of pAntler, which keeps track of which apps it has launched.

```
$ pAntler mission.moos
```



3

Killing the MOOS by Quitting pAntler

- pAntler can be killed by typing the "Control-C" character.
- As pAntler exit, it kills all the other applications it launched.

Mission Launched →

```
$ pAntler mission.moos
 -Iterate Mode 0 :
   |-Regular iterate and message delivery at 4 Hz

trying to open:[forrest19.tif]
Result:0x0
Trying to open: /Users/mikerb/Research/moos-ivp/branches/mikerb/moos-ivp-177bld/ivp/data/forrest19.tif
Success: /Users/mikerb/Research/moos-ivp/branches/mikerb/moos-ivp-177bld/ivp/data/forrest19.tif
set_pan_x:-90
set_pan_y:-280
zoom:0.65
.zoom:0.6175
..  Added wildcard logging of APPCAST_REQ_ALL
  Added wildcard logging of UPROCESSWATCH_ITER_GAP
.....  Added wildcard logging of APPCAST_REQ_ALPHA
.....
Ctrl-C
```

- Mission Output
- All applications writing their output to the same terminal window

Mission Killed →

4

A Dangling MOOS

- It is possible to close a Terminal window where pAntler was launched, *without killing pAntler*.
- A Dangling MOOS – The mission continues to run in the background unseen.

Run "top" command →

MOOS Apps reveal themselves

```
zeus:src/pMarineViewer(177bld)% top

Processes: 361 total, 2 running, 359 sleeping, 1334 threads                                  15:30:59
Load Avg: 2.24, 7.77, 12.91  CPU usage: 3.49% user, 2.28% sys, 94.21% idle
SharedLibs: 253M resident, 51M data, 75M linkedit.
MemRegions: 70659 total, 5712M resident, 179M private, 1355M shared.
PhysMem: 12G used (1927M wired), 4219M unused.
VM: 978G vsize, 633M framework vsize, 0(0) swapins, 0(0) swapouts.
Networks: packets: 2316229/1350M in, 2557994/2069M out. Disks: 802975/20G read, 412116/23G written.

PID    COMMAND      %CPU TIME     #TH   #WQ  #PORT MEM    PURG   CMPRS  PGRP  PPID  STATE
20979  top          3.3  00:00.70 1/1   0    20    4464K  0B     0B     20979 18918 running
20977  pEchoVar     0.2  00:00.04 6     0    24    908K   0B     0B     20977 20968 sleeping
20976  pNodeReporte 0.3  00:00.05 6     0    24    1008K  0B     0B     20976 20968 sleeping
20975  uProcessWatc 0.3  00:00.05 6     0    24    1084K  0B     0B     20975 20968 sleeping
20974  pMarineViewe 10.8 00:01.77 17    8    230   79M+   5536K  0B     20974 20968 sleeping
20973  pHelmIvP     0.5  00:00.07 6     0    24    1112K  0B     0B     20973 20968 sleeping
20972  pMarinePID   0.3  00:00.06 6     0    24    884K   0B     0B     20972 20968 sleeping
20971  uSimMarine   0.3  00:00.05 6     0    24    1116K  0B     0B     20971 20968 sleeping
20970  pLogger      0.9  00:00.14 6     0    24    1440K+ 0B     0B     20970 20968 sleeping
20969  MOOSDB       1.8  00:00.26 23    0    44    2424K  0B     0B     20969 20968 sleeping
20968  pAntler      0.0  00:00.01 1     0    10    372K   0B     0B     20968 17159 sleeping
20963  quicklookd   0.0  00:00.11 4     1    83    4096K  32K    0B     20963 1     sleeping
20952- mdworker32   0.0  00:00.64 3     1    51    5664K  0B     0B     20952 1     sleeping
```


5

Killing MOOS by Remote Killing of pAntler

- If the Terminal where pAntler was launched is no longer accessible (for a Ctrl-C Killing), a *remote kill* can be executed with the `killall` command:

Run "top" command →

MOOS Apps output

```
$ pAntler mission.moos
 -Iterate Mode 0 :
    |-Regular iterate and message delivery at 4 Hz

trying to open:[forrest19.tif]
Result:0x0
.........  Added wildcard logging of APPCAST_REQ_ALPHA   .............
```

At some point later this Terminal window closes w/out killing pAntler → ??

In a separate Terminal window, use the `killall` command →

```
$ killall  pAntler
$
```

Done. Mission should be killed. →

6

Killing MOOS by ktm

- Killing MOOS by killing pAntler only works for MOOS apps launched by pAntler.
- Other apps may be launched not using pAntler. (In separate terminal windows or in scripts).

From any Terminal window, invoke the `ktm` command →

```
$ ktm
```

- The `ktm` utility will find all MOOSDBs running and kill all apps connected to them.

Invoke `ktm` →

Will list all procesess killed

```
zeus:/Users/mikerb(177bld)% ktm
sending suicide instruction to 224.1.1.3:4000
pMarineViewer        pid 21167   on 100.73.205.120 is committing suicide
uProcessWatch        pid 21168   on 100.73.205.120 is committing suicide
pHelmIvP             pid 21166   on 100.73.205.120 is committing suicide
pMarinePID           pid 21165   on 100.73.205.120 is committing suicide
pLogger              pid 21163   on 100.73.205.120 is committing suicide
pEchoVar             pid 21170   on 100.73.205.120 is committing suicide
uSimMarine           pid 21164   on 100.73.205.120 is committing suicide
pNodeReporter        pid 21169   on 100.73.205.120 is committing suicide
MOOSDB_alpha         pid 21162   on 100.73.205.120 is committing suicide
zeus:/Users/mikerb(177bld)%
```

- Drawback: `ktm` works over multicast. It will not work if there is no network connection. It also apparently doesn't work with Window (WSL).

7

Killing MOOS by zkill

- The `zkill.sh` bash script uses `which` to find the bin directory for other MOOS app binaries.
- It then executes a `killall` for all binaries in that directory.

From any Terminal window, invoke the `zkill.sh` command →

```
$ zkill.sh MOOSDB -v
```

- The `zkill.sh` utility will send `killall` to all binaries in the same folder as the MOOSDB binary

Will list all processes killed when using the -v option

```
zeus:/Users/mikerb(177bld)% zkill.sh MOOSDB -v
Killing  MOOSDB  (pid: 36352)
Killing  pAntler  (pid: 36350)
Killing  pEchoVar  (pid: 36362)
Killing  pHelmIvP  (pid: 36356)
Killing  pLogger  (pid: 36353)
Killing  pMarinePID  (pid: 36355)
Killing  pMarineViewer  (pid: 36357)
Killing  pNodeReporter  (pid: 36361)
Killing  uMAC  (pid: 36351)
Killing  uProcessWatch  (pid: 36359)
Killing  uSimMarine  (pid: 36354)
zeus:/Users/mikerb(177bld)%
```

- Drawback: `zkill` will not work if MOOS binaries are installed in system folders, e.g., `/usr/bin/`

8

When the MOOS Can't Be Killed

Method 1: Killing the pAntler that launched the mission works every time, except:

- When the pAntler Terminal disappears and it's no longer possible to send Ctrl-C.
- When MOOS apps are connected that weren't launched by pAntler, e.g., uXMS, uMAC etc.

Method 2: Killing with `ktm` works every time, except:

- When there is no network interface. Or with perhaps with Windows WSL

Method 3: Killing with `zkill` works every time, except:

- When the MOOS binaries are installed in a system folder like `/usr/local/bin`.
- You also need to apply `zkill` to your own `moos-ivp-you/bin` folder.

9

Pratical Hacks for Killing the MOOS

- Each of the known methods for killing the MOOS have edge cases where they may miss an application, or just don't work.
- The hack is to do all of them – use an alias!

The `akill` alias

```
alias akill='ktm;    sleep 2;    killall -9 pAntler;    zkill.sh MOOSDB -v'
```

- Use the `ktm` utility
- Use the `sleep` utility
- Use the `killall` utility
- Use the `zkill` utility

10

Pratical Hacks for Killing the MOOS

- Each of the known methods for killing the MOOS have edge cases where they may miss an application, or just don't work.
- The hack is to do all of them – use an alias!

The `akill` alias

```
alias akill='ktm;     sleep 2;     killall -9 pAntler;    zkill.sh MOOSDB -v'
```

- Use the `ktm` utility
- Use the `sleep` utility
- Use the `killall` utility
- Use the `zkill` utility

- The alias can be augmented to include MOOS binaries as part of your `moos-ivp-you` tree.

```
alias akill='ktm; sleep 2; killall -9 pAntler; zkill.sh MOOSDB -v; zkill.sh pMyApp -v'
```


11

MOOS Top

- The command line (OSX or Linux) has the utility called "top"

The `top` command

```
$ top
```

MOOS Apps reveal themselves

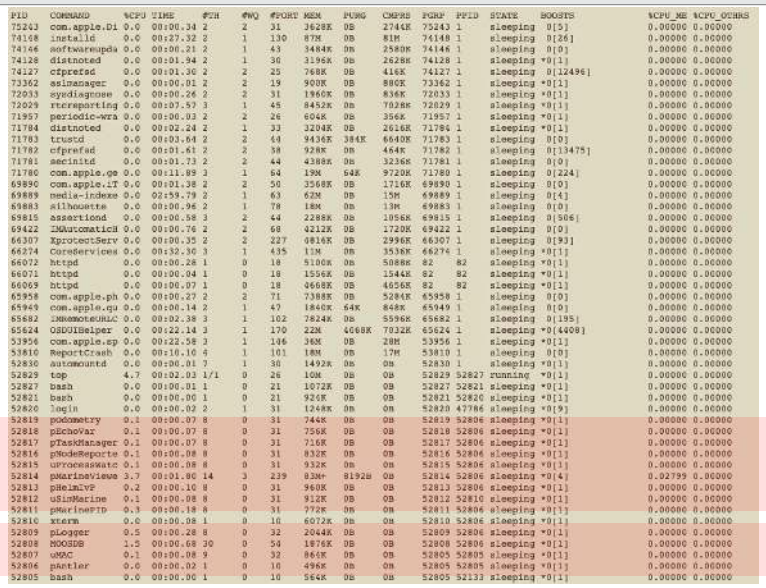
```
PID    COMMAND      %CPU TIME     #TH   #WQ  #PORT MEM    PURG   CMPRS  PGRP  PPID  STATE
20979  top          3.3  00:00.70 1/1   0    20    4464K  0B     0B     20979 18918 running
20977  pEchoVar     0.2  00:00.04 6     0    24    908K   0B     0B     20977 20968 sleeping
20976  pNodeReporte 0.3  00:00.05 6     0    24    1008K  0B     0B     20976 20968 sleeping
20975  uProcessWatc 0.3  00:00.05 6     0    24    1084K  0B     0B     20975 20968 sleeping
20974  pMarineViewe 10.8 00:01.77 17    8    230   79M+   5536K  0B     20974 20968 sleeping
20973  pHelmIvP     0.5  00:00.07 6     0    24    1112K  0B     0B     20973 20968 sleeping
20972  pMarinePID   0.3  00:00.06 6     0    24    884K   0B     0B     20972 20968 sleeping
20971  uSimMarine   0.3  00:00.05 6     0    24    1116K  0B     0B     20971 20968 sleeping
20970  pLogger      0.9  00:00.14 6     0    24    1440K+ 0B     0B     20970 20968 sleeping
20969  MOOSDB       1.8  00:00.26 23    0    44    2424K  0B     0B     20969 20968 sleeping
20968  pAntler      0.0  00:00.01 1     0    10    372K   0B     0B     20968 17159 sleeping
20963  quicklookd   0.0  00:00.11 4     1    83    4096K  32K    0B     20963 1     sleeping
20952- mdworker32   0.0  00:00.64 3     1    51    5664K  0B     0B     20952 1     sleeping
```


12

MOOS Top

- Sometimes finding the MOOS Apps is hard.
- If there are many other processes running on your computer.

MOOS Apps reveal themselves



13

MOOS Top

- The **mtop.sh** script is a utility for running to while filtering only on MOOS applications.
- It is in **`moos-ivp/scripts`** (which is probably already in your shell path)

The `top.sh` command

```
$ mtop.sh
```

Only MOOS Apps

PID	PGRP	PPID	%CPU	MEM	USER	COMMAND
52863	52863	52855	3.6	83M	mikerb	pMarineViewer
52857	52857	52855	1.6	1824K+	mikerb	MOOSDB
52858	52858	52855	0.6	1776K	mikerb	pLogger
52861	52861	52855	0.4	748K	mikerb	pMarinePID
52862	52862	52855	0.2	924K	mikerb	pHelmIvP
52864	52864	52855	0.1	864K+	mikerb	uProcessWatch
52868	52868	52855	0.1	736K	mikerb	pOdometry
52865	52865	52855	0.1	800K	mikerb	pNodeReporter
52867	52867	52855	0.1	728K	mikerb	pEchoVar
52866	52866	52855	0.1	736K	mikerb	pTaskManager

14

MOOS Top

```
$ mtop.sh
```

Running mtop.sh with no arguments will

- Look for the pAntler App,
- Find all apps having the pAntler PID as its Parent PID (PPID), and then
- Run top only on those processes.

Only MOOS Apps

PID	PGRP	PPID	%CPU	MEM	USER	COMMAND
52863	52863	52855	3.6	83M	mikerb	pMarineViewer
52857	52857	52855	1.6	1824K+	mikerb	MOOSDB
52858	52858	52855	0.6	1776K	mikerb	pLogger
52861	52861	52855	0.4	748K	mikerb	pMarinePID
52862	52862	52855	0.2	924K	mikerb	pHelmIvP
52864	52864	52855	0.1	864K+	mikerb	uProcessWatch
52868	52868	52855	0.1	736K	mikerb	pOdometry
52865	52865	52855	0.1	800K	mikerb	pNodeReporter
52867	52867	52855	0.1	728K	mikerb	pEchoVar
52866	52866	52855	0.1	736K	mikerb	pTaskManager

Potential drawback: If it wasn't launched with pAntler, it won't be listed.

15

MOOS Top

```
$ mtop.sh --apps=MOOSDB,pLogger,pOdometry
```

Running mtop.sh with the --apps argument will:

- Find the PIDs for all processing matching the apps in the list, and then
- Run top only on those processes.

Only

- MOOSDB
- pLogger
- pOdometry

PID	PGRP	PPID	%CPU	MEM	USER	COMMAND
52857	52857	52855	1.2	2504K	mikerb	MOOSDB
52858	52858	52855	0.5	3140K	mikerb	pLogger
52868	52868	52855	0.1	748K	mikerb	pOdometry

16

MOOS Top

Some useful command line switches:

```
$ mtop.sh -c
```

Is shorthand for:

```
$ mtop.sh --apps=pHelmIvP,MOOSDB,pMarineViewer,uSimMarine,pMarinePID,pLogger
```

```
$ mtop.sh --survey
```

- Will check for the presence of MOOS Apps and return 3 if none present, or 4 if there are any apps running
- Useful for using within bash scripts

17

Debugging and Logging

- Does it build?
- Does it crash?
- Does it receive the mail it should?
- Does it publish the mail it should?
- How to add and see debugging output on the terminal?
- Adding debugging output in the form of publications to the MOOSDB

18

Debugging: Does it Build?

Use the `which` command to see if it is built and findable

```
$ which pOdometry
/Users/you/moos-ivp-you/bin/pOdometry
```

If, instead of the above, you see:

```
$ which pOdometry
$
```

Then either (a) the app didn't build, or (b) it is not in your shell path

19

Debugging: Check your Shell Path

What to do if `$ which pOdometry` fails.

To check your shell path, go into the directory where the app is *supposed* to be built:

```
$ cd ~/moos-ivp-you/bin
$ ls
pExampleApp  pOdometry
```

If you ***DO*** see the app in the directory, then something is wrong with your shell path.
Take a look at the help page for setting your shell path:

http://oceanai.mit.edu/ivpman/help/cmdline_augment_shell_path

- If you ***DO NOT*** see the app in the bin directory, then either (a) the app didn't build, or (b) it built someplace else, in another directory somewhere.
- What to do: First check the output of your build. If you don't see any obvious errors, then do a system search on your computer for the binary. Find out where it was built.

20

Does Your Program Run?

- So presumably you have confirmed your program built, and is in the shell path.
- Does it run?
- Can you launch it without it crashing?

In two separate terminal windows, launch a MOOSDB and launch your app.

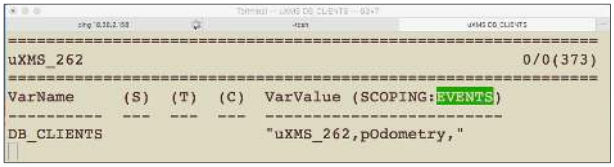
```
$ MOOSDB  file.moos
```

```
$ pOdometry  file.moos
```

If both seem to launch fine without quitting/crashing, then you have succeeded in the first milestone.

- You can also confirm with uXMS:

```
$ uXMS DB_CLIENTS
Enter IP address: [localhost]
Enter Port number: [9000]
```



21

Does Your Program Run *Continuously*?

- The test of durability is to launch your mission in the intended environment, processing the intended input.
- In the case of pOdometry, it will be processing **NAV_X** and **NAV_Y** mail messages

- What to do if:

A mission is launched → The mission seems to be running fine…. → At some point your app disappears / crashes.

```
$ pAntler alder.moos
```

No longer in the list of **DB_CLIENTS**

22

How to Debug a Disappearing App?

- The simplest way to debug an app that crashes is to check the terminal output just prior to crashing.
- You can add output upon entering OnNewMail() and Iterate() for example.
- Use your debugging output to narrow down the point of failure.

Problem: Under typical launch with pAntler, your app is not opened with its own terminal window. Even if you configure the Antler block in your mission file to open a window, this window disappears when the app disappears.

Solution: Remove the app that is crashing from the Antler block. Launch the mission without it. Then launch the crashing app in its own terminal window. This window will endure upon a crash and at least you can read any debugging output prior to the crash.

23

A Note about C++ and Debugging Output

- It is common to add simple debugging output in a program as a simple test to see where it crashes.

```
cout << "Got here 1" << endl;
function1();
cout << "Got here 2" << endl;
function2();
cout << "Got here 3";
function3();
cout << "Got here 4" << endl;
```

- If the last thing your program wrote to the terminal was:

```
Got here 1
Got here 2
```

- Then you know you should have a further look inside function2.

24

A Note about C++ and Debugging Output

- It is common to add simple debugging output in a program as a simple test to see where it crashes.

```
cout << "Got here 1" << endl;
function1();
cout << "Got here 2" << endl;
function2();
cout << "Got here 3";
function3();
cout << "Got here 4" << endl;
```

- If the last thing your program wrote to the terminal was:

```
Got here 1
Got here 2
```

- The problem may be in function2 or function3.

WARNING: Keep in mind that C++, output used in this way should be flushed to the terminal.

The use of **"<< endl"** accomplishes this.

Otherwise, if you want output without the newline, use **"<< flush"** explicitly.

A common gotcha in this kind of debugging is to wrongly infer the point of crash because there was pending output that was not flushed when the program crashed.

Better / More Complex Crash-Debugging

- Sending messages to the terminal is the simplest way to debug for new users.
- It can also be sufficiently productive, providing fast insight. Especially if you're quick with an editor and build.

Further ways to debug:
- Use a debugger, like **gdb**
- Create a core dump of your program crash
- Develop using an integrated development environment (IDE) rather than a text-editor.

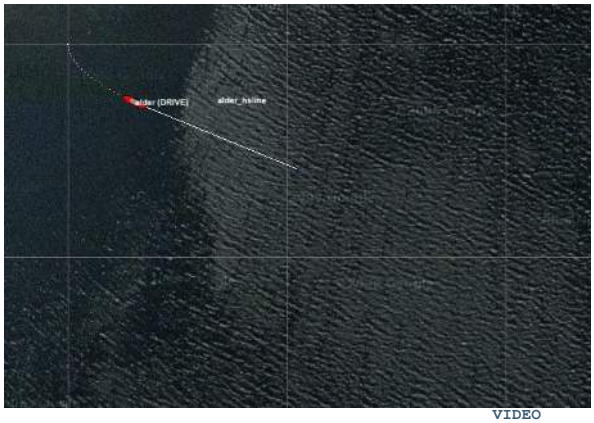
- The multiple apps running asynchronously in MOOS makes it trickier to use a debugger, stepping through.
- I very rarely see a crash that can't be debugged with terminal ouput.
- For the rare super tricky cases, a core dump usually provides all the extra insight needed.

Debugging Mission Correctness

- ✅ Your MOOS app builds.
- ✅ Your can find your MOOS app in your shell path.
- ✅ Your MOOS app does not crash upon launch.
- ✅ Your MOOS app never crashes even after the vehicle is underway, processing all intended mail.
- ❌ Mission executes as intended

Consider the Alder mission from Lab 4. Using the pOdometry app, it should return home after going 50 meters.

What's wrong?



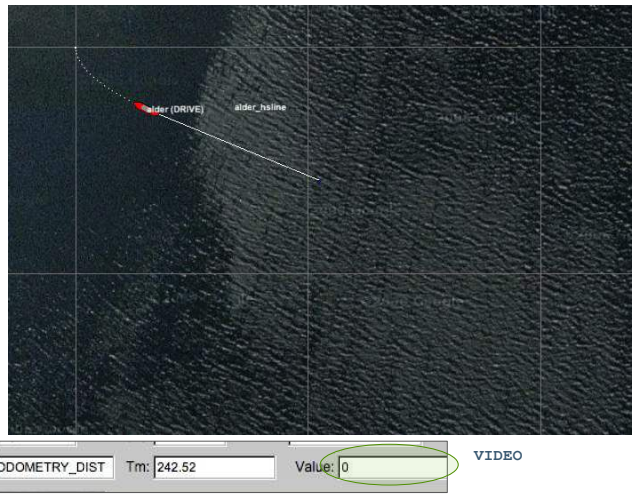
VIDEO

27

Debugging the Alder (pOdometry) Mission

Your first line of defense is to scope on a key variable or two and see if it is what you expect
In this mission, the key variable is `ODOMETRY_DIST`.
At first it seems correct as the vehicle launches.

- At first it seems correct as the vehicle launches.
- The distance grows at a plausible rate.
- But note that during the wiggle, the total distance oscillates above and below 50 meters.
- The odometry distance is incorrectly showing the current distance from the starting position.
- The helm is oscillating the return behavior on and off.



VIDEO

28

Debugging with the Logger

- The logger (pLogger) can reveal much more information.
- The first step is to enable it in your mission.

Step 1: Add it to your mission's Antler Block

```
ProcessConfig = ANTLER
{
  MSBetweenLaunches = 200


  Run = MOOSDB          @ NewConsole = false
  Run = uSimMarine      @ NewConsole = false
  Run = pNodeReporter   @ NewConsole = false
  Run = pMarinePID      @ NewConsole = false
  Run = pMarineViewer   @ NewConsole = false
  Run = uProcessWatch   @ NewConsole = false
  Run = pHelmIvP        @ NewConsole = false
  Run = pOdometryX      @ NewConsole = false
  Run = pLogger         @ NewConsole = false
}
```

Step 2: Configure pLogger. Add a config block on your mission file

```
ProcessConfig = pLogger
{
   AppTick   = 8
   CommsTick = 8

   AsyncLog  = true
   LogAuxSrc = true
   WildCardLogging = true
   WildCardOmitPattern = *_STATUS
}
```


29

Files Generated by the Logger

- After re-running the mission, there should now be a Log folder in your directory

```
$ ls
MOOSLog_21_2_2018_____17_27_51/
alder.bhv
alder.moos
```

- In this folder there should be a group of files similar to:

```
$ls
MOOSLog_21_2_2018_____17_27_51._moos
MOOSLog_21_2_2018_____17_27_51.xlog
MOOSLog_21_2_2018_____17_27_51.alog
MOOSLog_21_2_2018_____17_27_51.ylog
MOOSLog_21_2_2018_____17_27_51.blog
alder._bhv
```

- The **alog** file is of particular value.
- There are many tools in our toolbox (The "Alog Toolbox") for analysing these files.

30

Using the aloggrep Tool

- The aloggrep tool isolates one or more variables in an alog file:

```
$ aloggrep MOOSLog_21_2_2018_____17_27_51.alog ODOMETRY_DIST
138.923          ODOMETRY_DIST          pOdometryX      51.26181
139.616          ODOMETRY_DIST          pOdometryX      51.11645
139.848          ODOMETRY_DIST          pOdometryX      50.91528
140.068          ODOMETRY_DIST          pOdometryX      50.83589
140.551          ODOMETRY_DIST          pOdometryX      50.39295
140.768          ODOMETRY_DIST          pOdometryX      50.17305
141.220          ODOMETRY_DIST          pOdometryX      49.67764
141.440          ODOMETRY_DIST          pOdometryX      49.45527
141.855          ODOMETRY_DIST          pOdometryX      49.24637
142.097          ODOMETRY_DIST          pOdometryX      49.16875
142.541          ODOMETRY_DIST          pOdometryX      49.19578
143.033          ODOMETRY_DIST          pOdometryX      49.38980
143.279          ODOMETRY_DIST          pOdometryX      49.52313
143.759          ODOMETRY_DIST          pOdometryX      49.86080
144.190          ODOMETRY_DIST          pOdometryX      50.43569
144.653          ODOMETRY_DIST          pOdometryX      50.92384
144.891          ODOMETRY_DIST          pOdometryX      51.10971
145.104          ODOMETRY_DIST          pOdometryX      51.28451
```

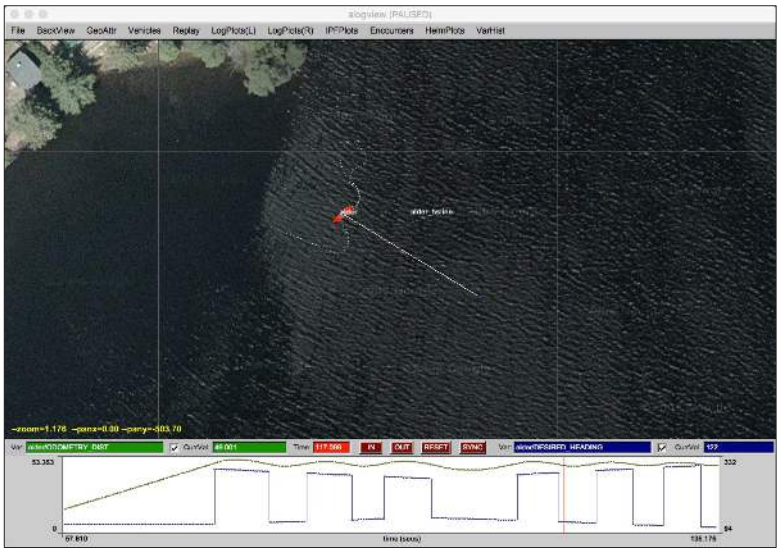
Even with this short segment, we can see that the odometry distance is *not* growing monotonically as it always should.

31

Using the alogview Tool

- The alogview enables you to replay a mission stored in one or more alog files:

```
$ alogview file.alog
```



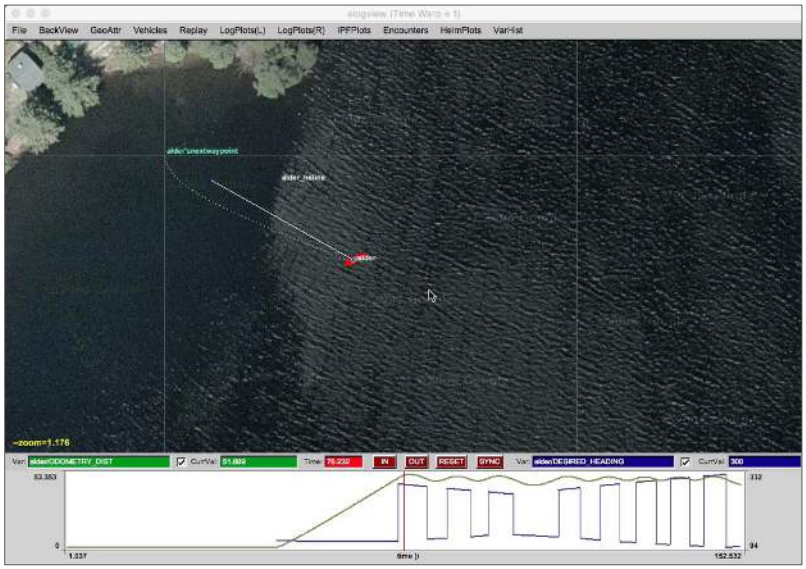
32

Using the alogview Tool

- The alogview enables you to replay a mission stored in one or more alog files:

```
$ alogview file.alog
```

Replay: toggle the spacebar key to start/stop the replay of the mission.



VIDEO

33

Terminal Output in alogview

- In 2022, a new feature was introduced to see terminal output within alogview

- Typically, if you are using terminal output to debug, the only way to do this is to launch your app in a separate terminal window and observe the output as it scrolls by.
- Once the mission has been quit, you can still scroll up the terminal window and observe the output, but it may be hard to visually tie the output you are seeing to where the vehicles were when the output was generated.

- Required steps:

1. Your app must be an AppCasting MOOSApp
2. In the config block of your app, in the mission file:

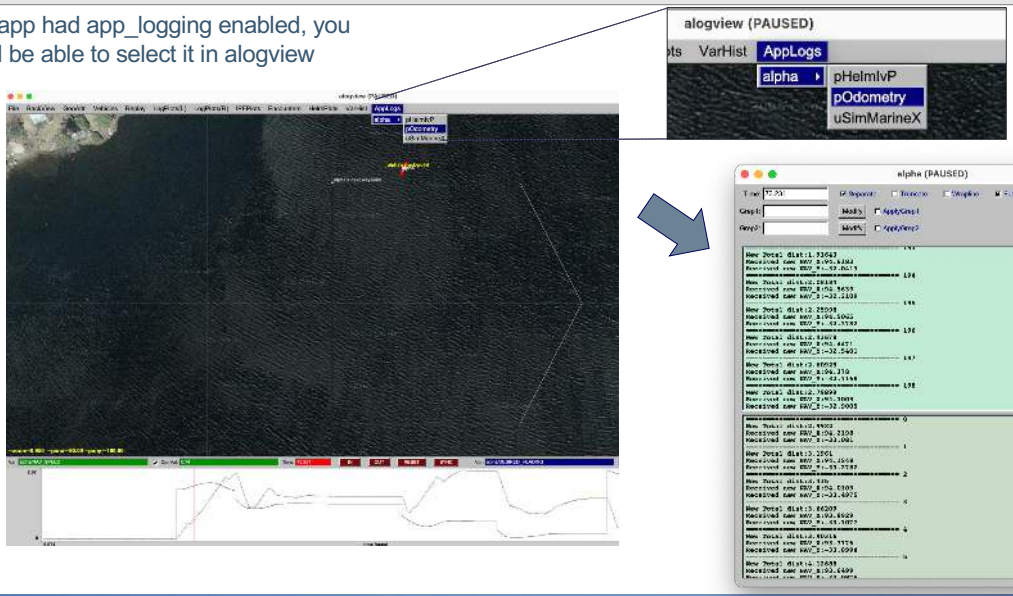
```
app_logging = true
```

3. Run the mission as normal. Quit. Go to the log folder.
4. Post-mission, run alogview Select AppLog from the pull-down menu Select the app you're looking for

34

Terminal Output in alogview

- If any app had app_logging enabled, you should be able to select it in alogview



35

MOOS-IvP Utilities

36

MOOS-IvP Utilities

- Both MOOS and IvP have many useful utilities
- Some utilities have two versions – one in MOOS and one in IvP.

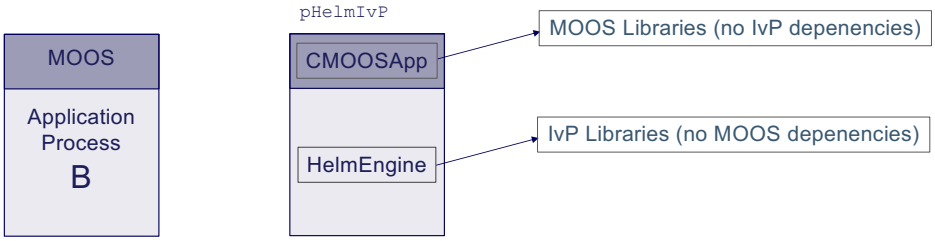
```
string MOOSToLower(string);
```

MOOS/MOOSCore/Core/libMOOS/Utils/include/
MOOS/libMOOS/Utils/MOOSUtilityFunctions.h

```
string tolower(string);
```

ivp/src/lib_mbutil/mbutils.h

Note: If you are using a MOOS App created with one of our scripts, then these utilities are likely already included by virtue of being a MOOS App.



37

String Case Conversion and Comparison

Case Conversion

```
string MOOSToLower(string);
string MOOSToUpper(string);
```

MOOS

```
string tolower(string);
string toupper(string);
```

IvP

```
string foo = "Hello World";
string bar = MOOSToLower(foo);
cout << "foo: " << foo << endl;
cout << "bar: " << bar << endl;
```

```
foo: Hello World
bar: hello world
```

Case Insensitive String Comparison

```
if(MOOSStrCmp(param, "AppTick"))
```

MOOS

```
if(tolower(param) == "apptick")
```

IvP

38

MOOS Time Utilities

Current Time

```
double MOOSTime(bool apply_warp=true);
```
MOOS

- Will return unix time, the number of seconds since January 1st 1970.
- If there is a time warp, it will apply the warp as a scalar multiple
- ***If you only use MOOSTime() in your app, your app will be time-warp enabled***.

```
double GetMOOSTimeWarp();
```
MOOS

- If you need to know the time warp from within your app, this will do the trick.
- Especially useful if an app is generating output for human live consumption, e.g. status updates.

```
double MOOSLocalTime(bool apply_warp=true);
```
MOOS

- Will return the time at the Comms Server (MOOSDB). An indicator of mission duration.

39

Stripping White Space

Stripping White Space

```
void MOOSTrimWhiteSpace(string&)
```
MOOS

*Note the value of the passed argument will be altered

```
string foo = " One, two, three ";
MOOSTrimWhiteSpace(foo);
cout << "foo: [" << foo << "]" << endl
```

```
foo: [One, two, three]
```

```
string stripBlankEnds(string);
```
IvP

```
string foo = " One, two, three ";
string bar = stripBlankEnds(foo);
cout << "foo: [" << foo << "]" << endl;
cout << "bar: [" << bar << "]" << endl;
```

```
foo: [ One, two, three ]
bar: [One, two, three]
```


40

String Bite/Chomp: Parsing from the Front

String Chomp Or Bite

`string MOOSChomp(string, string, bool)` MOOS

`string biteString(string, char);` IvP

`string rbiteString(string, char);` IvP

* Reverse biteString() just bites from the end instead.

Bite and Remove White Space

`string biteStringX(string, char);` IvP

* Short for `stripBlankEnds(biteString(string))`

```
string bar = "One, two, three";
string foo = biteString(bar, ',');
cout << "foo: " << foo << endl;
cout << "bar: " << bar << endl;
```

```
foo: One
bar:  two,three
```

```
string bar = "One, two, three";
string foo = biteStringX(bar, ',');
cout << "foo: " << foo << endl;
cout << "bar: " << bar << endl;
```

```
foo: One
bar: two,three
```


41

Parsing "Comma-Separated-Param-Value" Strings

- A common message type is the comma-separated parameter=value string

```
NODE_REPORT = NAME=alpha,TYPE=UUV,TIME=1252348077.59,X=51.71,Y=-35.50,
LAT=43.824981, LON=-70.329755,SPD=2.0,HDG=118.8, DEPTH=4.6,LENGTH=3.8,
```

Parsing Strings into Parts

`vector<string> parseString(string, char);` IvP

```
string str = "name=alpha, x=23, y=99";
vector<string> myvector = parseString(str, ',');

for(unsigned int i=0; i<myvector.size(); i++) {
  string param = biteStringX(myvector[i], '=');
  string value = myvector[i];

  if(tolower(param) == "alpha")
    handleAlpha(value);
  else if(tolower(param) == "x")
    handleX(value);
  else if(tolower(param) == "y")
    handleY(value);
}
```

* Typical structure for parsing a mail message

42

Converting Numbers to Strings

- To print out a number to the terminal, numbers don't need to be "converted" to strings

```
double temperature = 98.6;
cout << "The current tempature is " << temperature << endl
```

- To compose a MOOS string message with numerical components, some extra work is required.
- We'd like to do something simple like this:

```
double temperature = 98.6;
string message = "temperature=" + 98.6;      // Invalid syntax
Notify("HEALTH_STATUS", message);
```

- The STL stringstream is a common way to accomplish this (but a bit cumbersome):

```
#include <sstream>
stringsteam ss;
double value = 98.6;
ss << "temperature=" << value;
string message = ss.str();
Notify("HEALTH_STATUS", message);
```


Utilities for Converting Integers to Strings

Integers to Strings

```
string intToString(int);                    IvP
string uintToString(unsigned int);          IvP
string ulintToString(unsigned long int);    IvP
```

```
int weeks = 5;
string msg = "vacation" + intToString(weeks);
Notify("BENEFITS", msg);
```

Integers to Comma Strings

```
string intToCommaString(int);                    IvP
string uintToCommaString(unsigned int);          IvP
string ulintToCommaString(unsigned long int);    IvP
```

```
string msg = uintToCommaString(1234567);
cout << "Message: " << msg << endl;
```

```
Message = 1,234,567
```

Utilities for Converting Doubles and to Strings

Doubles to Strings

```
string doubleToString(double, int prec=5);
```
IvP

```
string doubleToStringX(double, int prec=5);
```
IvP

```
double val = 15.377;
string s1 = doubleToString(val,5);
string s2 = doubleToStringX(val,5);
string s3 = doubleToStringX(val,2);
cout << "Message 1: " << s1 << endl;
cout << "Message 2: " << s3 << endl;
cout << "Message 3: " << s3 << endl;
```

```
Message 1: 15.37700
Message 2: 15.377
Message 3: 15.38
```

- When converting strings to doubles, we typically use vanilla C or C++ utilities.

45

Utilities for Converting Booleans to/from Strings

- It is common to have MOOS variables hold Boolean values
- For example **DEPLOY=true** or **RETURN=false**, in the alpha mission and alder mission.
- These variables are actually just string values.
- We need to be careful that we don't inadvertently evaluate `("True" == "true")` to be false.

Booleans to Strings / Strings to Booleans

```
string boolToString(bool);
```
IvP

```
bool setBooleanOn(bool&, string);
```
IvP

```
bool isBoolean(string);
```
IvP

```
string s1 = "TRUe";
string s2 = "hello";
bool b0=true;
bool b1=false;
bool handled1 = setBooleanOnString(b1, s1);
bool b2=false;
bool handled1 = setBooleanOnString(b2, s2);

cout << "Message 1: " << isBoolean(s1) << endl;
cout << "Message 2: " << isBoolean(s2) << endl;
cout << "Message 2: " << boolToString(b0) << endl;
cout << "Message 4: " << b1 << "," << handled1 << endl;
cout << "Message 5: " << b2 << "," << handled2 << endl;
```

```
Message 1: 1
Message 2: 0
Message 2: true
Message 4: 1, 1
Message 5: 0, 0
```


46

Configuration Handling in MOOS Apps

In this section we will:

- Review the OnStartUp() method structure
- Review the kinds of misconfigurations
- Introduce a style for checking for unrecognized parameter names
- Introduce utilities for checking for invalide parameter values
- Advocate for AppCasting for raising warnings to operators

47

An Example OnStartUp() Method

- Here is the perfectly functional OnStartUp() function from the pXRelay app.
- We will walk through each line, and then offer additional important improvements.

```
bool Relayer::OnStartUp()
{
  STRING_LIST sParams;
  m_MissionReader.GetConfiguration(GetAppName(), sParams);

  STRING_LIST::iterator p;
  for(p = sParams.begin();p!=sParams.end();p++) {
     string line  = *p;
     cout << "[" << line << "]" << endl;
     string param = tolower(biteStringX(line, '='));
     string value = line;

     if(param == "incoming_var")
        m_incoming_var = value;
     else if(param == "outgoing_var")
        m_outgoing_var = value;
   }
   RegisterVariables();
   return(true);
}
```


48

Getting the Config Lines from the Mission File

```
0  bool Relayer::OnStartUp()
1  {
2    STRING_LIST sParams;
3    m_MissionReader.GetConfiguration(GetAppName(), sParams);
4    STRING_LIST::iterator p;
5    for(p=sParams.begin(); p!=sParams.end(); p++) {
6      string line  = *p;
7      string param = tolower(biteStringX(line, '='));
8      string value = line;
9
10     if(param == "incoming_var")
11       m_incoming_var = value;
12     else if(param == "outgoing_var")
13       m_outgoing_var = value;
14   }
15   RegisterVariables();
16   return(true);
17 }
```

[2] STRING_LIST is just an alias (typdef) for std::list<std::string>

[3] This MOOS app is a subclass of the general CMOOSApp superclass. The superclass has a member variable (m_MissionReader) of type CProcessConfigReader. This object is implemented to read the mission.moos file and return a list of strings holding all the config parameters for this app.

```
ProcessConfig = pXRelay
{
   AppTick   = 4
   CommsTick = 4
   outgoing_var = PEARS
   incoming_var = APPLES
}
```


Getting Config Lines - Under the Hood

- As an experiment, let's look at an OnStartUp() function that just prints its parameters to the terminal:

```
0  bool XRelay::OnStartUp()
1  {
2    STRING_LIST sParams;
3    m_MissionReader.GetConfiguration(GetAppName(), sParams);
4    STRING_LIST::iterator p;
5    for(p=sParams.begin(); p!=sParams.end(); p++) {
6      cout << "param: [" << *p << "]" << endl;
7    }
8    return(true);
9  }
```

```
ProcessConfig = pXRelay
{
   AppTick   = 4   // Really fast
   CommsTick = 4

   // This is a comment
   outgoing_var = PEARS
   incoming_var = APPLES
}
```

```
param: [incoming_var=APPLES]
param: [outgoing_var=PEARS]
param: [CommsTick=4]
param: [AppTick=4]
```

Notes:
- The order is reversed
- All white space has been removed
- Full line comments are removed
- Inline comments are removed
- Blank lines are removed

Getting Config Lines – White Space

Note:

- The `GetConfiguration()` call removes all white space – at the ends *and internally*.
- This can be a problem if the parameter value is a string with blank spaces.

```
0  bool XRelay::OnStartUp()
1  {
2    STRING_LIST sParams;
3    m_MissionReader.GetConfiguration(GetAppName(), sParams);
4    STRING_LIST::iterator p;
5    for(p=sParams.begin(); p!=sParams.end(); p++) {
6      cout << "param: [" << *p << "]" << endl;
7    }
8    return(true);
9  }
```

For Example:

```
ProcessConfig = pXRelay
{
   AppTick   = 4
   CommsTick = 4

   outgoing_var = PEARS
   incoming_var = APPLES
   message1     = Hello World
   message2     = "Goodbye World"
}
```

White space is gone, even when surrounded by quotes

```
param: [message2=GoodbyeWorld]
param: [message1=HelloWorld]
param: [incoming_var=APPLES]
param: [outgoing_var=PEARS]
param: [CommsTick=4]
param: [AppTick=4]
```


Getting Config Lines – Preserving White Space

Note:

- The `GetConfigurationAndPreserveSpace()` is an alternative with same arguments.
- Preserves white space internal to the param and value components.
- Produces a list in the same order as in the mission file. The other version reverses the order.

```
0  bool XRelay::OnStartUp()
1  {
2    STRING_LIST sParams;
3    m_MissionReader.GetConfigurationAndPreserveSpace(GetAppName(), sParams);
4    STRING_LIST::iterator p;
5    for(p=sParams.begin(); p!=sParams.end(); p++) {
6      cout << "param: [" << *p << "]" << endl;
7    }
8    return(true);
9  }
```

For Example:

```
ProcessConfig = pXRelay
{
   AppTick   = 4
   CommsTick = 4

   outgoing_var = PEARS
   incoming_var = APPLES
   message one  = Hello World
}
```

```
param: [AppTick=4]
param: [CommsTick=4]
param: [incoming_var=APPLES]
param: [outgoing_var=PEARS]
param: [message one=Hello World]
```

Notes:

- The order is *not* reversed
- White space internal to the parameter (text to the left of '=') is preserved.
- White space internal to the value (text to the right of '=') is preserved.
- Comments and blank lines a before.

White Space – To Preserve or Not?

- Use `GetConfiguration()`

  or

- Use `GetConfigurationAndPreserveSpace()`

Our Opinion:

- Best practice is to *make no assumptions about the order of parameters*. This just introduces another way for mission authors to make configuration errors. So the ordering difference between the two methods should be irrelevent.
- It is considered bad practice to have a configuration parameter ***name*** with white space, so the loss of white space to the left of the '=' is only important if you want to do something that's not recommended anyway.
- It does seem reasonable that a parameter *value* could have white space, e.g., specifying a message with white space. In this case you'll need the version that preserves white space. Except for this case, either method is fine to use.

Breaking Down Each Config Line Into Parts

```
0  bool XRelay::OnStartUp()
1  {
2    STRING_LIST sParams;
3    m_MissionReader.GetConfiguration(GetAppName(), sParams);
4    STRING_LIST::iterator p;
5    for(p=sParams.begin(); p!=sParams.end(); p++) {
6      string line  = *p;
7      string param = tolower(biteStringX(line, '='));
8      string value = line;
9
10     if(param == "incoming_var")
11       m_incoming_var = value;
12     else if(param == "outgoing_var")
13       m_outgoing_var = value;
14   }
15   RegisterVariables();
16   return(true);
17 }
```

[4][5] Using a vanilla C++ iterator to iterate through list<string> list of configuation lines.

[6] The config line is found by de-referencing the iterator p. I like to be clear and just copy the info into a variabe like "line" just to the code more obvious and readable.

[7] Get the parameter name (the text to the left of the '='. Convert it here to lower case to enable case insensitive comparisons below.

[8] Get the parameter value (the text to the right of the '='. It's just the leftover of "line" after biteString, but I like to be clear and just copy the info into a variabe like "value" just to the code more obvious and readable.

Handling Each Config Line – Case By Case

```
0  bool XRelay::OnStartUp()
1  {
2    STRING_LIST sParams;
3    m_MissionReader.GetConfiguration(GetAppName(), sParams);
4    STRING_LIST::iterator p;
5    for(p=sParams.begin(); p!=sParams.end(); p++) {
6      string line  = *p;
7      string param = tolower(biteStringX(line, '='));
8      string value = line;
9
10     if(param == "incoming_var")
11       m_incoming_var = value;
12     else if(param == "outgoing_var")
13       m_outgoing_var = value;
14   }
15   RegisterVariables();
16   return(true);
17 }
```

[10],[12] Case by case handle each known parameter type. Note each parameter *name* is matched against an all-lowercase string. This is in coordination with the tolower() call on line 7. By this we are implementing an app where the user is free to specify parameter names w/out regard to case. Thereby removing a possible configuration error if editing mission files directly.

[11],[13] Assign the value to the local member variable.

55

Final Steps of OnStartUp – Return Values

```
0  bool XRelay::OnStartUp()
1  {
2    STRING_LIST sParams;
3    m_MissionReader.GetConfiguration(GetAppName(), sParams);
4    STRING_LIST::iterator p;
5    for(p=sParams.begin(); p!=sParams.end(); p++) {
6      string line  = *p;
7      string param = tolower(biteStringX(line, '='));
8      string value = line;
9
10     if(param == "incoming_var")
11       m_incoming_var = value;
12     else if(param == "outgoing_var")
13       m_outgoing_var = value;
14   }
15   RegisterVariables();
16   return(true);
17 }
```

[15] Register for mail. In this case we're registering for whatever variable was assigned to m_incoming_var. Note in this app we need to wait to this point to register since what we're registering for wasn't known until the mission file has been read.

[16] If false is returned, the app will quit. Not possible here no matter how ill-formed the parameters may have been. In practice we'll want to do a some checking for parameter correctness and handle appropriately, perhaps returning false.

56

What to Do with a Bad Config Line?

- If the app is passed an unrecognized configuration parameter, this should be handled.
- Otherwise debugging a broken mission may require a lot of work, just to perhaps find a simple typo.

```
0  bool XRelay::OnStartUp()
1  {
2    STRING_LIST sParams;
3    m_MissionReader.GetConfiguration(GetAppName(), sParams);
4    STRING_LIST::iterator p;
5    for(p=sParams.begin(); p!=sParams.end(); p++) {
6      string line  = *p;
7      string param = tolower(biteStringX(line, '='));
8      string value = line;
9
10     if(param == "incoming_var")
11       m_incoming_var = value;
12     else if(param == "outgoing_var")
13       m_outgoing_var = value;
14     else if((param != "apptick") && (param != "commstick")) {
15       cout << "Unhandled parameter: " << param << endl;
16       return(false);
17     }
18   }
19   RegisterVariables();
20   return(true);
21 }
```

[14] Check for an unrecognized parameter. If the param is not a valid for this app, also have to check if it's a param defined at the CMOOSApp superclass level.

[15] It's helpful if we tell the user a bit more about the problem. At least identify the unrecognized parameter.

[16] By returning false, app will quit.

Debate:
- Do we really want to quit the app?
- If this app is not launched in its own enduring terminal, the error output may never be seen.
- May be better to continue the launch and raise a config warning to the user. How?
- We'll use AppCasting for this!

57

Parameter Correctness Checking

- A good startup method should check for legal params names *and* legal param values.
- Determine if a param value is legal

```
0  bool XRelay::OnStartUp()
1  {
2    STRING_LIST sParams;
3    m_MissionReader.GetConfiguration(GetAppName(), sParams);
4    STRING_LIST::iterator p;
5    for(p=sParams.begin(); p!=sParams.end(); p++) {
6      string line  = *p;
7      string param = tolower(biteStringX(line, '='));
8      string value = line;
9
10     bool handled = false;
11     if(param == "param1")
12       handled = setParam1(value);
13     else if(param == "param2")
14       handled = setParam2(value);
15     else if((param == "apptick") || (param == "commstick"))
16       handled = true;
17
18     if(!handled)
19       cout << "Unhandled config: " << *p << endl
20   }
21   RegisterVariables();
22   return(true);
23 }
```

[10] Assume a parameter is unhandled unless shown otherwise.

[11][13] Each parameter handling switch invokes a routine which (a) returns a Boolean indicating the legality of the parameter value, and (b) actually sets the member variable if legal value.

[15] If the parameter name is defined at the CMOOSApp superclass level, just go ahead and mark it as handled.

[18-19] For now deal with an unhandled parameter by producing terminal output warning. We'll want to do this with AppCasting instead if building an AppCasting app.

58

Checking if a String represents a Number

- Utility functions enable us to do better correctness checking (and more concisely).
- A common check is to determine if a numerical parameter value is actually numerical.

Problem: String to number conversion routines like `atof` and `atoi` typically return 0 if the string is not numerical:

```
string value1 = "0";
string value2 = "a lot";
double dval1 = atof(value1.c_str());
double dval2 = atof(value2.c_str());
cout << "Value 1 is " << dval1 << endl;
cout << "Value 2 is " << dval2 << endl;
```

```
Value1 is 0
Value2 is 0
```

It's impossible to tell whether the original value was a legit zero or an erroneous string.

Solution: A utility function that checks if a string is numerical or not.

```
bool isNumber(string, bool blanks_ok=true);
```
IvP

Evaluates **true**

```
isNumber("0");
isNumber("-3.22")
isNumber("+98")
isNumber("   8")
```

Evaluates **false**

```
isNumber("one");
isNumber("34 ", false)
isNumber("1.95718e+09"
isNumber("2,3")
```


59

Setting a Parameter that Must be a Double

```
bool isNumber(string);
```
IvP

```
bool isBoolean(string);
```
IvP

The **`isNumber()`** and **`isBoolean()`** utilities enable further super useful utilities

```
bool setDoubleOnString(double&, string);
```

Returns true if the string argument is a number

Sets the given double (passed by reference) if the string argument is a number. Leaves it alone otherwise.

The string argument is the parameter value taken from the mission file.

```
bool handled = false;
if(param == "any_number")
  handled = setDoubleOnString(m_number, value);

if(!handled)
  cout << "Unhandled config: " << *p << endl
```

The jobs of (a) checking the legality of the parameter value, (b) assigning it to the member variable if valid, and (c) returning the result as a Boolean, can all be done with a single line.

60

Other Type-Specific Parameter Setting

Other similar utilities:

```
bool setDoubleOnString(double&, string);
bool setPosDoubleOnString(double&, string);
bool setNonNegDoubleOnString(double&, string);
bool setBooleanOnString(bool&, string);
bool setNonWhiteVarOnString(string&, string);
```

* Useful when referring to the name of a MOOS variable where no white spaces are wanted.

Concise parameter handling with:

- Parameter correctness testing
- Conditional assignment
- Rejection of unknown parameters
- Reporting to the user of anything wrong

```
bool handled = false;
if(param == "any_number")
  handled = setDoubleOnString(m_any_number, value);
else if(param == "alive")
  handled = setBooleanOnString(m_alive, value);
else if(param == "age")
  handled = setNonNegDoubleOnString(m_age, value);
else if(param == "moos_var")
  handled = setNonWhiteVarOnString(m_moos_var, value);

if(!handled)
  cout << "Unhandled config: " << *p << endl
```


61

Complex Type-Specific Parameter Setting

What to do when the parameter being set is not one of these simple types?

```
bool setDoubleOnString(double&, string);
bool setPosDoubleOnString(double&, string);
bool setNonNegDoubleOnString(double&, string);
bool setBooleanOnString(bool&, string);
bool setNonWhiteVarOnString(string&, string);
```

If the correctness check is simple, just handle it directly:

```
if((param == "station") && ((value=="AM")||(value="FM")))
  m_station = value;
  handled = true;
}
```

If it is more complex, handle it with a dedicated method:

```
if((param == "polygon_search_region")
  handled = setConvexPolygonSearchRegion(value);
```


62

What To Do with a Bad Parameter?

- Rule number 1: Prevent a misconfiguration from doing harm.
- Rule number 2: Make sure it is brought to the attention of the user/operator.

Prevent a misconfiguration from doing harm

- A mis-configuration could cause the vehicle to simply not run, or do something unintentional immediately, or (perhaps worse) do something unintential at some point mid-mission.
- A well-written app should bound the range or choices of input, but using default or bounded values upon mis-configuration could still result in unintended consequence.
- In some cases, quitting the app upon misconfiguration is appropriate. Not typically.

Make sure a misconfiguration is brought to the attention of the user/operator

- Writing terminal output may be useful, but depends on a terminal being open and watched.
- Posting debug information to the MOOSDB may also help. It assumes some other app is doing something with, or scoping, on this information.
- Our preferred method is AppCasting and using the `reportConfigWarning()` utility.

63

AppCasting Configuration Warnings

AppCasting MOOS apps use the following utility for reporting configuration warnings:

```
void reportConfigWarning(string);
```

```
bool handled = false;
if(param == "any_number")
  handled = setDoubleOnString(m_any_num, value);
else if(param == "alive")
  handled = setBooleanOnString(m_alive, value);
else if(param == "age")
  handled = setNonNegDoubleOnString(m_age, value);

if(!handled)
  cout << "Unhandled config: " << *p << endl
```

Instead of simple terminal output, the warning message is reported for AppCasting.

```
bool handled = false;
if(param == "any_number")
  handled = setDoubleOnString(m_any_num, value);
else if(param == "alive")
  handled = setBooleanOnString(m_alive, value);
else if(param == "age")
  handled = setNonNegDoubleOnString(m_age, value);

if(!handled) {
  string msg = param + "=" + value;
  reportConfigWarning("Unhandled param: " + msg);
}
```

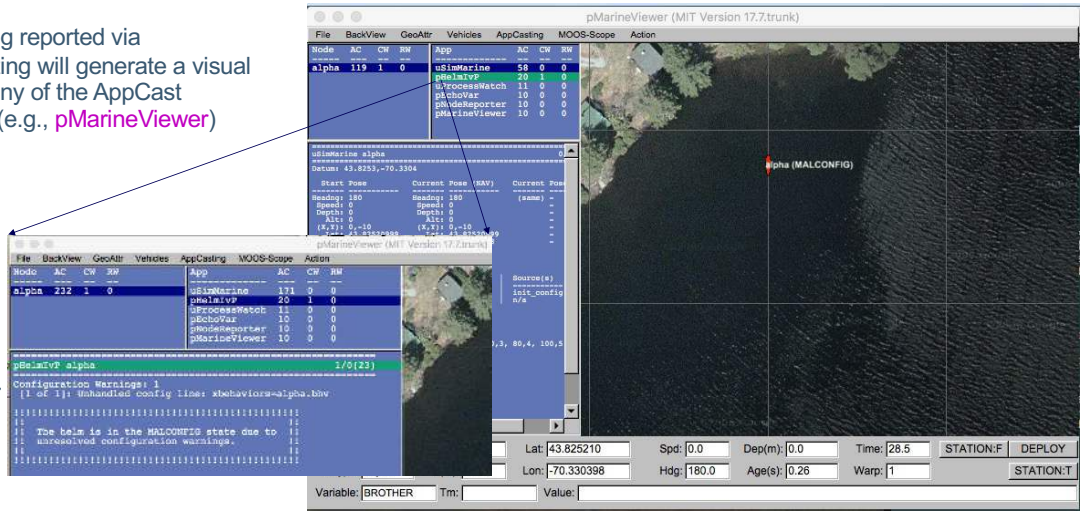
(with AppCasting, the warning will *also* go to the terminal)

64

Catching Configuration Warnings

A warning reported via AppCasting will generate a visual alert in any of the AppCast viewers (e.g., pMarineViewer)

Warning and offending line



65

Handling Run Warnings in MOOS Apps

In this section we will:

- Review the OnNewMail() method structure
- Review the kinds of run warnings
- Advocate for AppCasting for raising run warnings to operators
- How to revoke run warnings when appropriate

66

An Example OnNewMail() Method

- Here is the perfectly functional OnNewMail() function from the pOdometry app.
- We will walk through key lines, and then offer additional important improvements.

```
0  bool Odometry::OnNewMail(MOOSMSG_LIST &NewMail)
1  {
2     MOOSMSG_LIST::iterator p;
3     for(p=NewMail.begin(); p!=NewMail.end(); p++) {
4       CMOOSMsg &msg = *p;
5       string key   = msg.GetKey();
6       double dval  = msg.GetDouble();
7
8       if(key == "NAV_X")
9         m_current_x = dval;
10      else if(key == "NAV_Y")
11        m_current_y = dval;
12    }
13    return(true);
14 }
```

[0] All mail registered for by this app will be sent by the MOODB into this app's inbox, as an argument to the OnNewMail() function.

[2] MOOSMSG_LIST is an alias (typdef) for std::list<CMOOSMsg>

[4-6] This app only registers for **NAX_X** and **NAV_Y**. So we never look for string values, just grab the double value with msg.GetDouble().

[8-11] This app only registers for **NAX_X** and **NAV_Y**. So we only handle these two mail types.

[13] This app always returns true. But returning false will not result in any discernable difference… Just FYI.

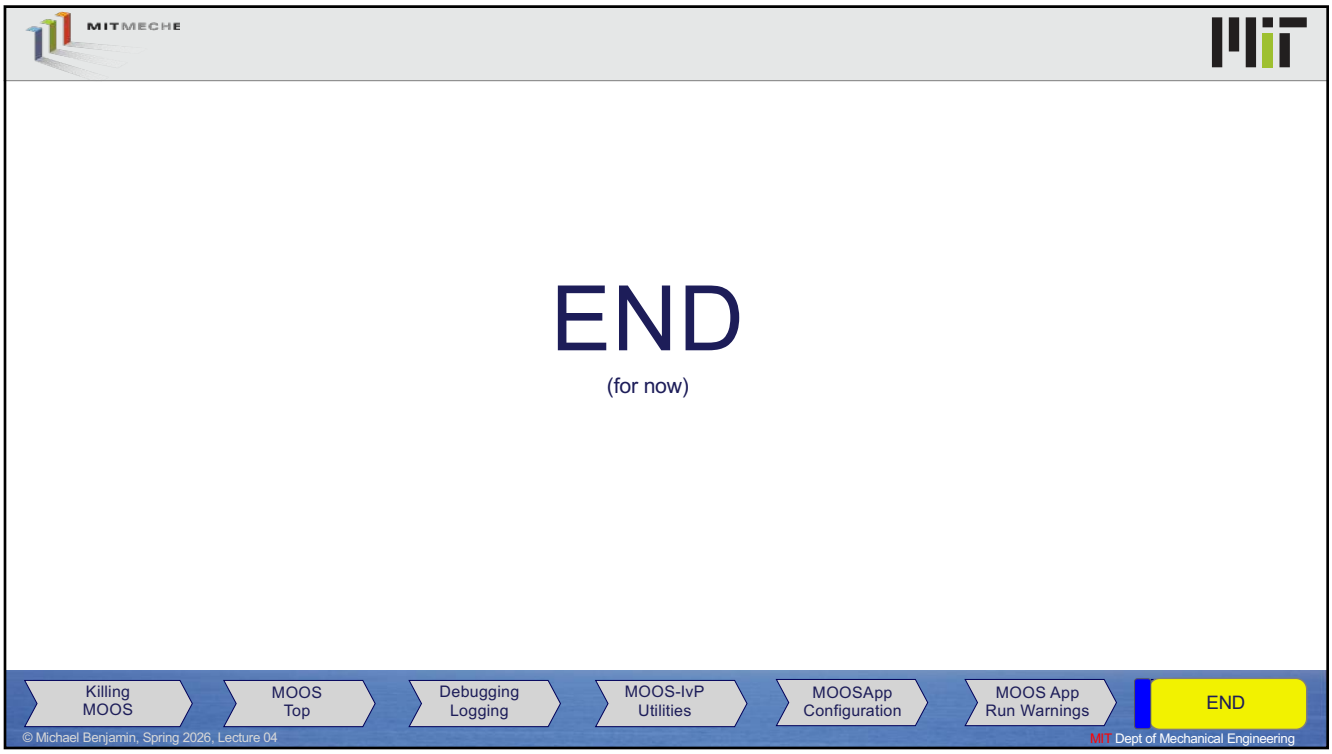
67

When Mail Handling is Simple

- For the pOdometry app, this is perfectly fine mail handling.
- Only two variables, **NAV_X** and **NAV_Y** are registered for. Unlike parameter handling, we will never receive mail of an unexpected or mis-typed variable name.
- Both member variables (m_current_x and m_current_y) are of a simple double type. There's no conceivable "wrong" value for incoming mail. No error checking needed.

```
0  bool Odometry::OnNewMail(MOOSMSG_LIST &NewMail)
1  {
2     MOOSMSG_LIST::iterator p;
3     for(p=NewMail.begin(); p!=NewMail.end(); p++) {
4       CMOOSMsg &msg = *p;
5       string key   = msg.GetKey();
6       double dval  = msg.GetDouble();
7
8       if(key == "NAV_X")
9         m_current_x = dval;
10      else if(key == "NAV_Y")
11        m_current_y = dval;
12    }
13    return(true);
14 }
```


68


MITMECHE
END
(for now)
Killing MOOS
MOOS Top
Debugging Logging
MOOS-IvP Utilities
MOOSApp Configuration
MOOS App Run Warnings
END
© Michael Benjamin, Spring 2026, Lecture 04
MIT Dept of Mechanical Engineering

69