Launching MOOS Processes and Mission Scripts with pAntler

Paul Newman, Oxford

1 Overview **1**

2 Basic Syntax **2**

3 Controlling Process Launch **2**
- 3.1 Launching Processes in new console windows (or not) . . . 2
- 3.2 Controlling Console Appearance . . . 2
- 3.3 Appearance Example . . . 3
- 3.4 Controlling Search Paths . . . 4

4 Passing Parameters to Launched Processes **4**
- 4.1 The Two Default Parameters . . . 4
- 4.2 Handling default parameters . . . 5

5 Running Multiple Instances of a Particular Process **5**
- 5.1 Customising the Command Line Parameters Passed to a Launched Process . . . 5
- 5.2 Specifying Additional Process Command Line Parameters . . . 5
- 5.3 Inhibiting default parameters and Launching Arbitrary (non-MOOS) Processes . . . 6

6 Distributing a Community over Multiple Machines **6**
- 6.1 Motivation . . . 6
- 6.2 Antler Modes: Monach and Headless . . . 7

7 Examples **8**
- 7.1 Local Configurations . . . 9
- 7.2 Distributed Configuration . . . 11

8 Application note : I/O Redirection - Deployment **12**

1 Overview

This section discusses how to use the application `pAntler` to launch multiple MOOS processes. This is useful tool for starting up a set of processes all of which share a single configuration file. The process `pAntler` is used to launch/create a MOOS community. It is simple to use and Post V7.0.2 very extensible. One of the ideas underlying MOOS is the one mission file one mission paradigm. A single mission file contains all the information required to configure all the processes needed to undertake the task (mission) in hand. (The `pLogger` application backs up each mission file so the user can discern exactly what mission file was run at the time data was recorded - this is described in the `pLogger` documentation). Note a collection of MOOS processes is commonly referred to as a *community*.

2 Basic Syntax

pAntler provides a simple and compact way to start a MOOS mission. For example if the desired mission file is Mission.moos then the following would launch the required processes/community for the mission:

```
pAntler Mission.moos
```

It reads from its configuration block *(which is declared as* ProcessConfig=ANTLER*)* a list of process names that will constitute the MOOS community. Each process to be launched is specfied with a line with the general syntax:

```
Run = procname [ @ LaunchConfiguration] [~ MOOSName]
```

where LaunchConfiguration is an optional comma separated list of "parameter=value" pairs which collectively control how the process "procname" (for example iGPS, or iRemote or MOOSDB) is launched. Exactly what parameters can be specified is detailed later in the document. pAntler looks through its entire configuration block and launches one process for every line which begins with RUN=. When all processes have been launched pAntler waits for all of them to exit and then quits itself.

3 Controlling Process Launch

Immediately after the "@" symbol in a RUN directive the user can supply a list of "parameter=value" pairs (comma separated) which control how the process in question should be launched. The following subsections will explain the action of available parameters.

3.1 Launching Processes in new console windows (or not)

```
Run = MOOSDB  @NewConsole = true
```

The optional NewConsole parameter specifies whether the named process should be launched in a new window (an xterm in Unix or cmd-prompt in Win32 derived platforms). By default a new console is launched.

3.2 Controlling Console Appearance

Post V7.0.2 releases allow a good deal of control over the appearance of the windows in which processes will be launched. Especially so on the 'nix side of life. (The native win32 console has less flexibility than the xterm. Deep apologies for Win32 users who may feel hard done by by the asymmetry here.)

By specifying XConfig=<Name> or Win32Config=<Name> (depending on OS) the user can have pAntler apply customisations to the new console in a process is launched. For example:

```
Run = MOOSDB  @NewConsole = true,XConfig=DBXConsoleSettings,  \
                            Win32Config=DBW32ConsoleSettings
```

will cause pAntler to search through its configuration block to find a line which begins with DBXConsoleSettings =. or DBW32ConsoleSettings = – depending on OS). What is the left of the equality determines the appearance of the new console and is a function of the host operating system.

Console Appearance in Unix like OS's

In unix derived operating systems the appearance string (referenced by "XConfig") is a comma separated list of parameters that would be used to configure an xterm. So continuing by way of the DBConsoleSettings example. If the DBConsoleSettings was specified (on its own line) as follows

```
DBXConsoleSettings = -bg,#FF0000,-fg,#FFFFFF,-geometry,80x12+2+00,+sb,-T,TheMOOSDB
```

then the MOOSDB would be launched in 12 rows by 80 columns window, white text on red at the top left of teh screen. The string "TheMOOSDB" would appear in the title. Note any xterm configuration parameters can specified in this way. See the manual page for xterm for information on the options allowed.

Console Appearance in Win32 OS's

The only native WIN32 console options supported control the background color of the terminal (text is always white). The LHS of the configuration line (referenced by "Win32Config") can contain a comma separated list of BACKGROUND_RED BACKGROUND_BLUE and BACKGROUND_GREEN. The following would produce a white on red win32 console:

```
DBW32ConsoleSettings = BACKGROUND_RED
```

3.3 Appearance Example

Listing 3.1: An example Antler configuration block.

```
1  //------------------------------------------------------------------
2  // pLogger configuration block
3  ProcessConfig = Antler
4  {
5    // look on system path
6    ExecutablePath = system
7
8    // launch a DB
9    Run = MOOSDB  @NewConsole = true,XConfig=DBXConsoleSettings,Win32Config=DBW32ConsoleSettings
10
11   // xterm configuration for DB
12   DBXConsoleSettings = -bg, \#FF0000,-fg,\#FFFFFF,-geometry,80x12+2+00,+sb,-T,TheMOOSDB
13
14   // Win32 Configuration for DB
15   DBW32ConsoleSettings = BACKGROUND_RED
16 }
```

3.4 Controlling Search Paths

Post V7.0.2 pAntler offers extended functionality regarding specifying how executables are located on the host file system. The paths which you wish the OS to use when searching for executable to launch can be specified globally (a common path for all processes) or on a process by process basis.

Specifying Global Executable Paths

Adding line of the form

```
ExecutablePath = path
```

to Antler's configuration block where *path* is a suitable path string, will make pAntler search in that place for the exectables to launch. Not specifying this variable or setting path to "SYSTEM" will cause Antler to relie on the host OS being able to locate the executable in its own executable paths.

Specifying Paths for an Individual Process

The global executable path (default "system") can be overridden for a particular process by providing your prefered path in the "RUN" directive line. For example:

```
Run = pP1 @  NewConsole = true,path=/usr/strangeplace
```

will try to launch a process called "pP1" from a the directory "/usr/strangeplace". Such process specific path directives overide any path set with `ExecutablePath=...` (Section 3.4).

4 Passing Parameters to Launched Processes

4.1 The Two Default Parameters

Unless told otherwise (see 5.3) each process launched is passed the mission file name as a command line argument and also the name it should use to register with the MOOSDB. This means that by default `argv[1]` of `main` is the name of the mission file currently in play (the one which pAntler is itself reading) and argv[2] is the name of the process in to be launched (for example iGPS or pLogger). By default pAntler assumes the name which a process will be registering with the MOOSDB with is the name of the process itself. For example pLogger will register with the MOOSDB with the name "pLogger". However this can be changed using the `MOOSName` syntax:

```
Run = iGPS  @NewConsole = true ~ GPS_A
```

The above will cause the executable iGPS to be launched in a new console but (because iGPS handles command line parameters appropriately) it will register with the MOOSDB under the name of "GPS_A".

4.2 Handling default parameters

Of course just passing the MOOSName to a process doesn't mean automagically that all MOOS connections within that process will use this name. Supporting code must be provided.

5 Running Multiple Instances of a Particular Process

As already described in Section 4, the optional `MOOSName` parameter allows MOOSProcesses to connect to the `MOOSDB` under a specified name. Why is this useful? Well for example a vehicle may have two GPS instruments onboard. Now by default `iGPS` may register it existence with the `MOOSDB` under the name `iGPS`. This name is now taken and no other MOOSClient can use the name "iGPS" - if they try the MOOSDB will not accept them into the fold. By using the syntax multiple instances of the executable `iGPS` can be run but with each connecting to a the `MOOSDB` using a different name. For example:

```
Run = iGPS @ NewConsole = true ~iGPSA
Run = iGPS @ NewConsole = true ~iGPSB
```

would launch two instances of `iGPS` registering under "iGPSA" and "iGPSB" respectively. Note there would need to be two GPS configuration blocks in the mission file – one for each and the process names (RHS of `ProcessConfig=`) would be "iGPSA" and "iGPSB"

5.1 Customising the Command Line Parameters Passed to a Launched Process

But what if your beloved new process which you desire `pAntler` to launch requires extra command line configuration? Or what if you don't want `pAntler` to pass the Mission file name and the MOOS name in a parameters? Fear not, just read on.

5.2 Specifying Additional Process Command Line Parameters

You can specify additional parameters which should be passed to a launched process using a syntax similar to that used to specify console appearance (see Section 3.2) The trick is to specify the name of a parameter string (R.H.S of a `ExtraProcessParams=...` in the process's `RUN` directive line. `pAntler` the rescans its configuration block looking for this named string which must be a comma separated list of parameters. An example will make this blindingly obvious.

```
ProcessConfig = Antler
{
  Run = iProcA @ NewConsole = true,path=/usr/local/bin, ExtraProcessParams=ProcAParams
  ProcAParams =-o,--verbose,--clever
}
```

The above would launch a process called `iProcA` in a new console, (with default appearance as no appearance string is specified see Section 3.2), and the process will be passed *six* parameters at launch time:

1. `argv[0]`: the executable image name.

2. argv[1]: the mission file name
3. argv[2]: the process's MOOS name
4. argv[3]: -o
5. argv[4]: –verbose
6. argv[5]: –clever

5.3 Inhibiting default parameters and Launching Arbitrary (non-MOOS) Processes

If you want to launch a process with pAntler that has not been designed to handle the mission file name an MOOS name as the first two parameters passed in the command line then it is possible to tell pAntler not to pass these parameters. This is done using the InhibitMOOSParams key word. For example if you wanted to launch the executable top in its own window you would use a configuration similar to:

```
Run = top @ InhibitMOOSParams=true, NewConsole=true
```

6 Distributing a Community over Multiple Machines

6.1 Motivation

Up until now we have implicitly assumed that all processes launched by a single instance of pAntler reside on the same physical computer. Surely this is conflicts with the idea that any MOOS process can run on any machine under any (common) OS? You're right it does and this issue has been addressed in post V7.0.2 versions. Excellent. In the broadest of terms it is possible to have have one Antler send a single mission file to a host of other Antlers (presumably but not necessarily sitting on a different machine or OS) which they then process and launch processes locally. The idea is that you still only need to edit one mission file to control a suite of processes running over any number of physical machines. The operating paradigm is that once a suitably configured pAntler has been started on a machine you need never kill or restart it. It stays alive patiently waiting for a instructions. See Figure 1

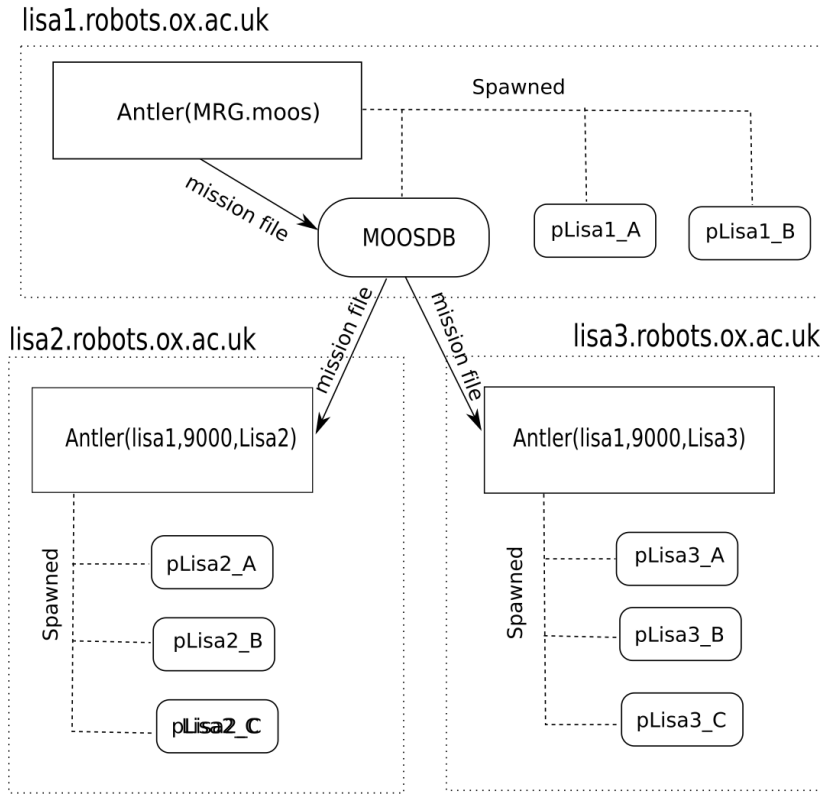

Figure 1: In distributed mode, pAntler can be started in one of two ways. Here on the machine lisa1.robots.ox.ac.uk it is started in "Top MOOS" mode with the name of a mission file on the command line. On the two other machines (lisa2 and lisa3) Antler is started in headless mode receiving three command line parameters - the machine name on which a MOOSDB can be found, the port that MOOSDB is serving on and an AntlerID name which in this case is simply set to the machine name. When the "topMOOS" has spawned its processes it pushes the mission file to the DB. The headless Antlers pick up this notification and run themselves from the newly received Mission file. Each headless Antler only launches processes which have a Run directive line containing that particular instantiation of Antler's ID.

6.2 Antler Modes: Monach and Headless

The idea is that pAntler can be run in one of two modes which we shall refer to as "headless" and "monach" (as in Monach of the Glen - referring to the size antlers). These terms only have meaning if the EnableDistributed flag is set to `true` in the Antler configuration block – ie when Antler is being told to support distributed process control. If this flag is set and pAntler is launched in the usual way:

```
./pAntler Mission.moos
```

then this will become a "Monach". Think of it as king/governing/top/controlling pAntler which will take responsibility for distributing (via the MOOSDB) the mission file to any other "headless" Antlers sitting on other machines. If however you start Antler with three command line parameters as follows

```
./pAntler lisa1.robots.ox.ac.uk 9000 lisa2
```

then Antler will launch in "headless" mode. Headless `Antlers` are bound to a single "Monachs" via a MOOSDB (which will usually be lauched by the monach itself). The three parameters specify the location and port of this MOOSDB and alls an `AntlerID`. This last parameter is a string which is used by headless Antlers to figure out which Run directives they should execute. Consider the following simple example:

```
ProcessConfig = Antler
{
  EnableDistributed = true
  Run = iProcA @ AntlerID = lisa2, NewConsole = true
  Run = MOOSDB @ NewConsole=true
}
```

Note how iProcA has an `AntlerID` specified. Now if as above one started a headless pAntler with "lisa2" as its Antler ID on machine "B" and then started another instance of pAntler on machine "B". (A can equal B but whats the point?) but this time only specifying a mission file (ie start pAntler as a "monach") you would witness a MOOSDB coming up on machine A and iProcA starting on machine B. If no `AntlerID` is specified in a run directive, it is assumed that the monach is required to process the directive. Headless Antlers only process run directives possessing an `AntlerID` matching their own. Each headless Antler writes the runtime received mission file (stripped of comments and superfluous white space) to local disk (working directory) under the name `dynamic_<TIMESTAMP>.moos` for future perusal.

Shutdown Behaviour

The default behaviour is for headless Antlers to shut down all their spawned processes when contact is lost with the MOOSDB. If this is not the desired behaviour and you want launched processes to carry on running simply add the directive `"KillOnDBDisconnect=false"` to the configuration block.

```
ProcessConfig = Antler
{
  EnableDistributed = true
  KillOnDBDisconnect=false
  Run = iProcA @ AntlerID = lisa2, NewConsole = true
  Run = MOOSDB @ NewConsole=true
}
```

In any case as soon as a Mission file is received by a headless Antler any and all running processes will be shutdown before processing the new Mission file.

7 Examples

If you enable the building of examples via the CMake build screen (See Figure 2) then the example configurations in Sections 7.1 and 7.2 serve as a good starting point in experimenting with pAntler. There are three examples processes supplied in the sibling code directory of the documentation:

1. `pAnterTestAppA` is nothing more than a dumb CMOOSApp that prints a string declared in its configuration block
2. `pAnterTestAppB` is nothing more than a dumb CMOOSApp which takes more than the standard two command line arguments, it uses these additional params to publish a variable to a MOOSDB
3. `pAnterTestAppC` is not a CMOOSApp. its just a program which prints out its command line arguments and spins in a do nothing loop.

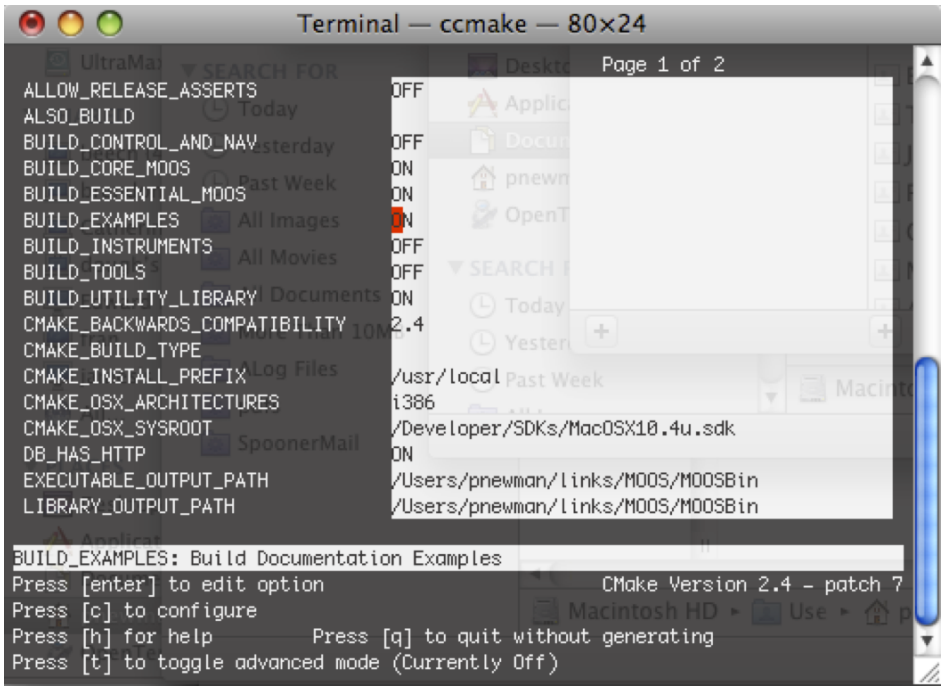

Figure 2: selecting the building of examples in the MOOS build screen

7.1 Local Configurations

Listing 7.2: Example Configuration Blocks for Antler where all process are run on the same host machine..

```
// Un-Comment/Comment the first line of each each example block
// to play with various Antler configurations

// Simplest possible example
// ProcessConfig = Antler
{
  Run = MOOSDB @ NewConsole = true
  Run = pAntlerTestAppA @ NewConsole = true
}

// Run two instances of pAntlerTestAppA under different names
// note two new configuration blocks are needed (Oxford and FenTech)
// ProcessConfig = Antler
```

```
{
  Run = MOOSDB @ NewConsole = true
  Run = pAntlerTestAppA @ NewConsole = true  ~ Oxford
  Run = pAntlerTestAppA @ NewConsole = true ~ FenTech
}


// Passing an additional two parameters to pTestAppB
// ProcessConfig = Antler
{
  Run = MOOSDB @ NewConsole = true
  Run = pAntlerTestAppA @ NewConsole = true

  Run = pAntlerTestAppB @ ExtraProcessParams = BParams , NewConsole = true
  BParams = CustomVar,ThisIsAString
}


// Specifying a default executable path and overloading it for MOOSBD
// ProcessConfig = Antler
{
  ExecutablePath  = C:/codescratch/MOOS/MOOSBin/debug/q
  Run = MOOSDB @ path=C:/codescratch/MOOS/MOOSBin/debug, NewConsole = true
  Run = qq @ NewConsole = true
}


// Passing three parameters to pTestAppC which is not expecting the first
// two parameters to be Mission File and MOOSName
// ProcessConfig = Antler
{
  Run = MOOSDB @ NewConsole = true

  Run = pAntlerTestAppA @ NewConsole = true

  Run = pAntlerTestAppB @ ExtraProcessParams = BParams , NewConsole = true
  BParams = CustomVar,ThisIsAString

  Run = pAntlerTestAppC @ ExtraProcessParams = CParams, InhibitMOOSParams=true, NewConsole = true
  CParams = set,the,moos,loose,1,2,3,45.6
}

// Adding some colour to MOOSDB, pAntlerTestB and pAntlerTestC
// ProcessConfig = Antler
{
  Run = MOOSDB @Win32Config=DBWin32,XConfig=DBX , NewConsole = true
  DBX = -bg,#FF0000, -geometry, 80x40+200+300
  DBWin32 = BACKGROUND_RED

  Run = pAntlerTestAppA @ NewConsole = true

  Run = pAntlerTestAppB @ Win32Config=BWin32,XConfig=BX, ExtraProcessParams = BParams, \
        NewConsole = true
  BParams = CustomVar,ThisIsAString
  BWin32 = BACKGROUND_GREEN,BACKGROUND_BLUE
```

```
  BX = -bg,#00FFFF, -geometry, 80x40+350+300

  Run = pAntlerTestAppC @ Win32Config=CWin32,XConfig=CX,ExtraProcessParams = CParams, InhibitMOOSParams=true , Ne
  CParams = set,the,moos,loose,1,2,3,45.6
  CWin32 = BACKGROUND_RED,BACKGROUND_BLUE
  CX = -bg,#FF00FF, -geometry, 80x40+400+300
}


// Configuration for TestAppA - just looks for a string to print
ProcessConfig = pAntlerTestAppA
{
  PrintThis = SetTheMOOSLoose
}

// Configuration for pTestAppB  - nothing but it expects a third and fourth
// command line tell it what to publish...
// ProcessConfig = pAntlerTestAppB
{

}

// Configuration for FenTech (which is actually  and instantiation of
// pAntlerTestAppA) - just looks for a string to print
ProcessConfig = FenTech
{
  PrintThis = ThisIsTestAppAAsFenTech
}

// Configuration for Oxford (which is actually  an instantiation of
// pAntlerTestAppA) - just looks for a string to print
ProcessConfig = Oxford
{
  PrintThis = ThisIsTestAppAAsOxford
}
```

7.2 Distributed Configuration

Listing 7.3: Example Configuration Blocks for Antler where all processes are run on different hosts.

```
 1  // If you are really running this on different hosts make sure you
 2  // set the server hostname below if you are simply testing how to run
 3  // multipls instances of MOOS leaving it as local host is just fine
 4
 5  ServerHost = localhost
 6  ServerPort = 9000
 7
 8  // Adding some colour to MOOSDB, pAntlerTestB and pAntlerTestC
 9  ProcessConfig = Antler
10  {
11    EnableDistributed = true
12    Run     = MOOSDB @ Win32Config=DBWin32,XConfig=DBX , NewConsole=true
13    DBX     = -bg,#FF0000, -geometry, 80x40+200+300
14    DBWin32 = BACKGROUND_RED
```

```
15
16    Run     = pAntlerTestAppA @ AntlerID=jupiter, NewConsole=true
17
18    Run     = pAntlerTestAppB @ AntlerID=neptune, Win32Config=BWin32, \
19              XConfig=BX, ExtraProcessParams=BParams , NewConsole=true
20    BParams = CustomVar, ThisIsAString
21    BWin32  = BACKGROUND_GREEN,BACKGROUND_BLUE
22    BX      = -bg,#00FFFF, -geometry, 80x40+350+300
23
24    Run     = pAntlerTestAppC @ Win32Config=CWin32, XConfig=CX, \
25              ExtraProcessParams=CParams, InhibitMOOSParams=true , NewConsole=true
26    CParams = set, the, moos, loose, 1, 2, 3, 45.6
27    CWin32  = BACKGROUND_RED,BACKGROUND_BLUE
28    CX      = -bg, #FF00FF, -geometry, 80x40+400+300
29  }
30
31  // Configuration for TestAppA - just looks for a string to print
32  ProcessConfig = pAntlerTestAppA
33  {
34    PrintThis = SetTheMOOSLoose
35  }
36
37  // Configuration for pTestAppB  - No params here, but it expects
38  // a third and fourth command line argument, a variable and string.
39  ProcessConfig = pAntlerTestAppB
40  {
41  }
42
43  // Configuration for pTestAppC  - No params here, but it expects
44  // eight additional command line arguments
45  ProcessConfig = pAntlerTestAppB
46  {
47  }
```

8 Application note : I/O Redirection - Deployment

Frequently iRemote, displayed on a remote machine, will be the only interface a user has to the MOOS community. We must ask the question - "where does all the IO from other processes go to prevent I/O blocking?". One answer to this is I/O redirection and backgrounding MOOS processes - a simple task in unix derived systems (Some OS are good for development others for running.) Running pAntler in the following fashion followed by a manual start up of iRemote is the recommended way of running MOOS in the field on a 'nix platform.

```
./pAntler mission.moos > ptyZ0 > /dev/null &
./iRemote mission.moos
```

This redirection of iRemote is encapsulated in the moosbg script included with the MOOS installations. In the case of an AUV the interface can only be reached through in-air wireless communications, which will clearly disappear when the vehicle submerges but will gracefully re-connect when surfacing (not so easy to do with a PPP or similar link).