uFleetManager Guide

Raphael Segal

June 21, 2017

1 Purpose

This guide is intended to explain to a member of the PAVLAB how to use and improve the fleet manager app. Those members are expected to have a basic familiarity with MOOS-IvP, C++, and Bash.

2 Usage

2.1 Dependencies

`uFleetManager` was developed for Mac. It is in principle compatible with Linux, but that has never been demonstrated.

Currently, the only dependency is `ncurses`. On a Mac, get it with either Macports or Homebrew;

```
port install ncurses
```

The usage is slightly more complicated on Linux. Without having gotten it working, it's hard to say for sure, but it looks like `libncurses5-dev` is the correct version. So on Ubuntu, get it with

```
apt-get install libncurses5-dev
```

2.2 Installation

`uFleetManager` is bundled in the `moos-ivp-aquaticus` tree. Assuming you haven't already, install `moos-ivp-aquaticus` in your home directory.

2.2.1 Download ARO

Most users will use the Anonymous Read Only version of `moos-ivp-aquaticus`:

```
svn co https://oceanai.mit.edu/svn/moos-ivp-aquaticus-aro-trunk/trunk
moos-ivp-aquaticus
```

2.2.2 Download for Editing

A few users will have edit and commit privileges; speak to Dr. Benjamin. Then get the codebase with

```
svn co svn+ssh://[YOUR_USERNAME]@oceanai.mit.edu/home/svn/repos/moos-ivp-aquaticus/trunk
moos-ivp-aquaticus
```

2.2.3 Enable

At the time of writing, this code is considered experimental and therefore disabled by default.

Open `~/moos-ivp-aquaticus/src/CMakeLists.txt` and in the `BUILD_ALL` section, find the line `ADD_SUBDIRECTORY(uFleetManager)` and uncomment it. Remember to recomment it before committing code, and check it after pulling down new code.

2.3 Starting the Fleet Manager

Build the fleet manager with the normal aquaticus build script, e.g.:

```
~/moos-ivp-aquaticus/build.sh
```

You'll be looking for the `uFleetManager` executable in `~/moos-ivp-aquaticus/bin`

There are two use cases; with or without a config file. You can run `uFleetManager` with no arguments, e.g.

```
~/moos-ivp-aquaticus/bin/uFleetManager
```

and that will let you observe all the lab vehicles but wont let you launch a MOOS-IvP mission.

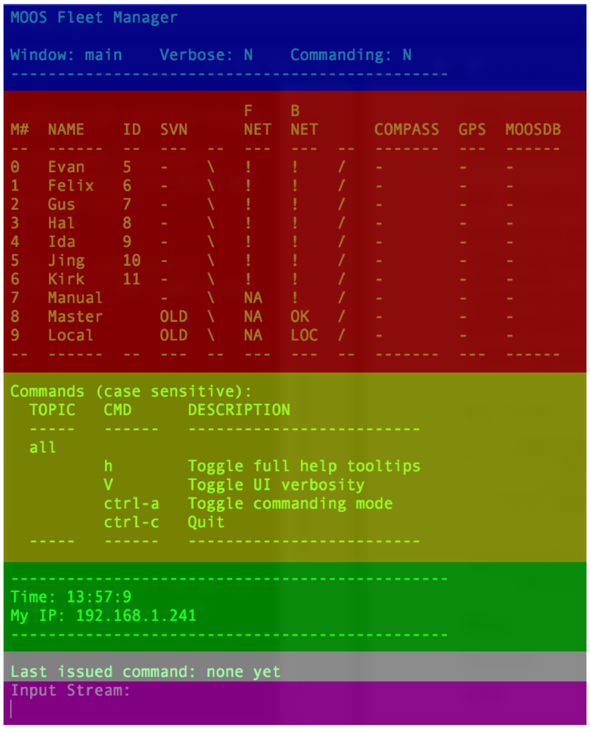
To only observe specific vehicles, and to run a particular MOOS-IvP mission, you'll want to run it with a config file;

```
~/moos-ivp-aquaticus/bin/uFleetManager --file /path/to/my_config_file.moos
```

Refer to the Config Files section for how to write a configuration file.

2.4 Fleet Maanger Layout

Once you start the Fleet Manager, you should see something like this, without the highlight colors.



```
MOOS Fleet Manager

Window: main    Verbose: N    Commanding: N
-----------------------------------------------

                         F    B
M#  NAME    ID  SVN      NET  NET           COMPASS  GPS  MOOSDB
--  ------  --  ---  --  ---  ---  --  -------  ---  ------
0   Evan    5   -    \   !    !    /   -        -    -
1   Felix   6   -    \   !    !    /   -        -    -
2   Gus     7   -    \   !    !    /   -        -    -
3   Hal     8   -    \   !    !    /   -        -    -
4   Ida     9   -    \   !    !    /   -        -    -
5   Jing    10  -    \   !    !    /   -        -    -
6   Kirk    11  -    \   !    !    /   -        -    -
7   Manual      -    \   NA   !    /   -        -    -
8   Master      OLD  \   NA   OK   /   -        -    -
9   Local       OLD  \   NA   LOC  /   -        -    -
--  ------  --  ---  --  ---  ---  --  -------  ---  ------

Commands (case sensitive):
  TOPIC   CMD      DESCRIPTION
  -----   ------   -------------------------
  all
          h        Toggle full help tooltips
          V        Toggle UI verbosity
          ctrl-a   Toggle commanding mode
          ctrl-c   Quit
  -----   ------   -------------------------

-----------------------------------------------
Time: 13:57:9
My IP: 192.168.1.241
-----------------------------------------------

Last issued command: none yet
Input Stream:
```

Blue Header; displays the state of the app

Red Window; displays the view indicated in the header (see below for details on each view).

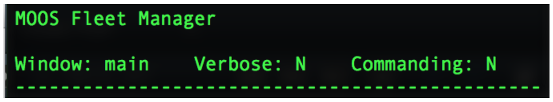
Yellow Help; displays the currently available command set.

Green My Machine; displays own computer information. 'Time' is the one topic that is expected to change frequently and consistently, and therefore can be used to determine if the app has crashed.

Grey Last command; displays the last command issued.

Purple Input; shows currently input characters

2.4.1 Header



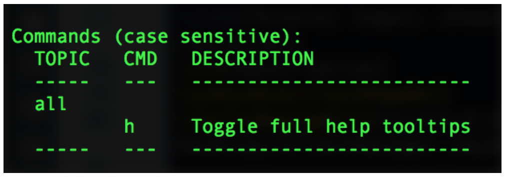
```
MOOS Fleet Manager

Window: main    Verbose: N    Commanding: N
-----------------------------------------------
```

Displays three state variables: the current view, whether verbosity is toggled on or off, and whether commanding is toggled on or off. Sections with multiple levels of verbosity have an asterisk after their headers, and will be noted below.

2.4.2 Windows

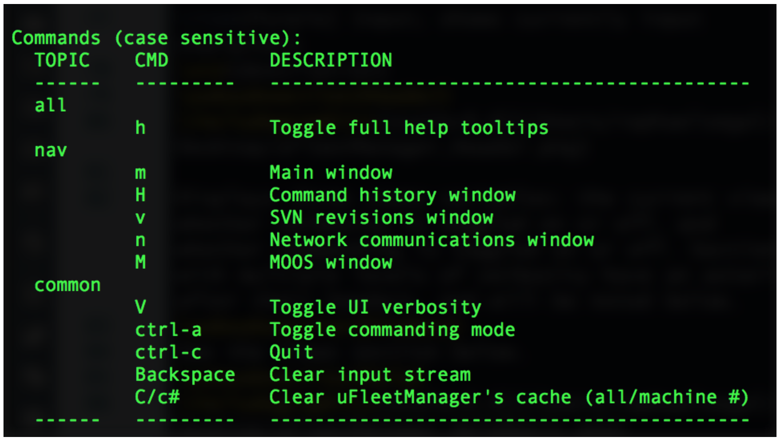
See the Views section below.

2.4.3 Help



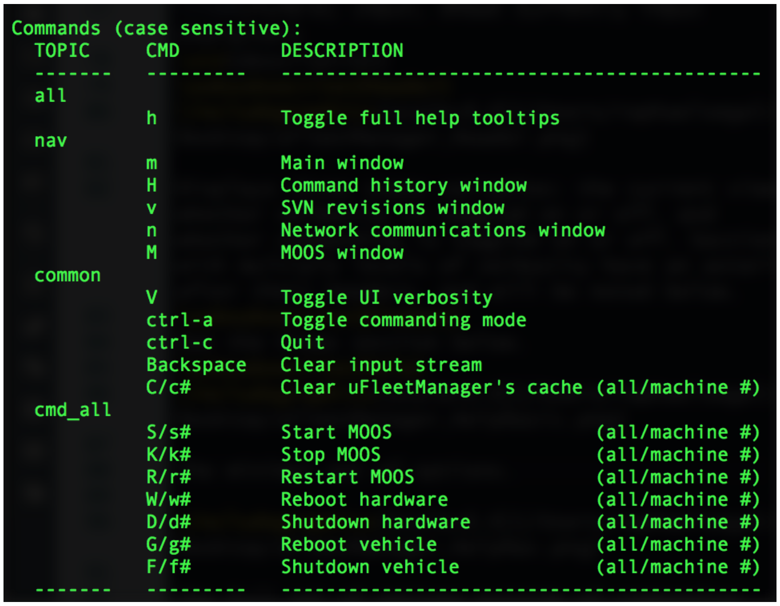
```
Commands (case sensitive):
  TOPIC   CMD   DESCRIPTION
  -----   ---   ------------------------
  all
          h     Toggle full help tooltips
  -----   ---   ------------------------
```

The minimal set of options.



```
Commands (case sensitive):
  TOPIC    CMD         DESCRIPTION
  ------   ---------   -------------------------------------------
  all
           h           Toggle full help tooltips
  nav
           m           Main window
           H           Command history window
           v           SVN revisions window
           n           Network communications window
           M           MOOS window
  common
           V           Toggle UI verbosity
           ctrl-a      Toggle commanding mode
           ctrl-c      Quit
           Backspace   Clear input stream
           C/c#        Clear uFleetManager's cache (all/machine #)
  ------   ---------   -------------------------------------------
```

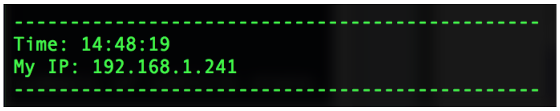
The full set of options outside of commanding mode; contains the common set, navigation commands, and the command to clear the local cache of information requests.



```
Commands (case sensitive):
  TOPIC     CMD         DESCRIPTION
  -------   ---------   -------------------------------------------
  all
            h           Toggle full help tooltips
  nav
            m           Main window
            H           Command history window
            v           SVN revisions window
            n           Network communications window
            M           MOOS window
  common
            V           Toggle UI verbosity
            ctrl-a      Toggle commanding mode
            ctrl-c      Quit
            Backspace   Clear input stream
            C/c#        Clear uFleetManager's cache (all/machine #)
  cmd_all
            S/s#        Start MOOS                  (all/machine #)
            K/k#        Stop MOOS                   (all/machine #)
            R/r#        Restart MOOS                (all/machine #)
            W/w#        Reboot hardware             (all/machine #)
            D/d#        Shutdown hardware           (all/machine #)
            G/g#        Reboot vehicle              (all/machine #)
            F/f#        Shutdown vehicle            (all/machine #)
  -------   ---------   -------------------------------------------
```

The full set of options, including those in commanding mode.

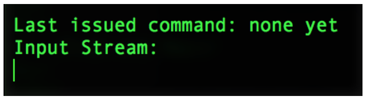
2.4.4 My Machine



```
-----------------------------------------------
Time: 14:48:19
My IP: 192.168.1.241
-----------------------------------------------
```

Stats about your own machine. `Time` serves as a responsive UI element, demonstrating that the app is actually refreshing. `MY IP` is helpful if you're running the shoreside on your computer and need to update the machine's UI, but it also indicates which wifi network you're on; if the first block is 10 you're probably on MIT-GUEST, if the wifi is 192.168.1.X, you're probably on kayak-local. You will only be able to talk to the robots on kayak-local.

2.4.5 Footer



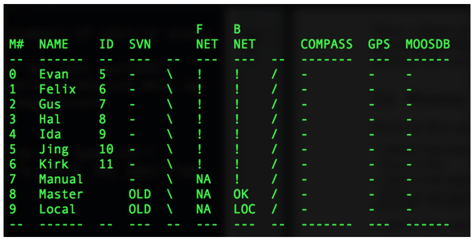
```
Last issued command: none yet
Input Stream:
|
```

Information about keys you're currently inputting, and the executive summary of the command you've most recently input.

2.5 Views

View Name	Nav Key	Description
Main	m	Main window, provides a ready/not ready summary of vehicle state.
Network	n	Vehicle addresses and whether ping and ssh test succeed
SVN Revisions	v	Lists revisions and summarizes which trees are most up-to-date for `moos-ivp`, `moos-ivp-aquaticus`, `moos-ivp-colregs`, `pablo-common`, and `mokai-common`
Command History	H	Lists the commands dispatched by the operator
MOOS-IvP	M	Lists mission configuration and details about the specified mission
Previous	p	Go to previous view

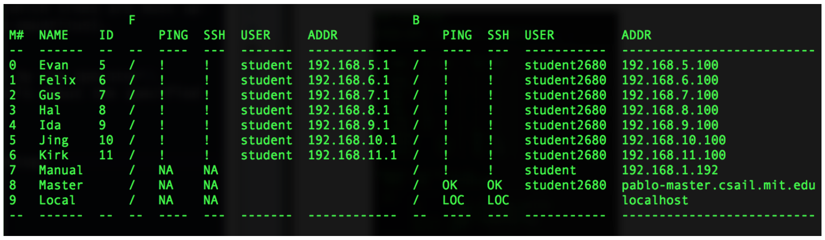
2.5.1 Main



```
                         F    B
M#  NAME    ID  SVN      NET  NET       COMPASS  GPS  MOOSDB
--  ------  --  ---  --  ---  ---  --  -------  ---  ------
0   Evan    5   -    \   !    !    /   -        -    -
1   Felix   6   -    \   !    !    /   -        -    -
2   Gus     7   -    \   !    !    /   -        -    -
3   Hal     8   -    \   !    !    /   -        -    -
4   Ida     9   -    \   !    !    /   -        -    -
5   Jing    10  -    \   !    !    /   -        -    -
6   Kirk    11  -    \   !    !    /   -        -    -
7   Manual      -    \   NA   !    /   -        -    -
8   Master      OLD  \   NA   OK   /   -        -    -
9   Local       OLD  \   NA   LOC  /   -        -    -
--  ------  --  ---  --  ---  ---  --  -------  ---  ------
```

Topic	Explanation	Comments
M#	Machine #; the # in the Commands section	Limited to 0 <= $M\#$ < 10
Name	Vehicle Name	List hard coded in Configuration class
ID	Lab vehicle id system, alpha=1, bravo=2, ...	
SVN	Summary; worst status from all its svn trees	OLD and NEW are relative amongst vehicles e.g. if even one of your trees is out of date, then your summary will be OLD. See SVN view for more detail
F NET	Front Seat network summary	ssh and ping; see Network view for more detail
B NET	Back Seat network summary	same as F NET. Single-computer robots are back seats
COMPASS	Reports if vehicle's compass is up	M300 common failure mode is NaNs Mokai common failure mode is disconnects
GPS	Reports GPS status	M300 reports PDOP Mokai only reports connectedness
MOOSDB	Counts the MOOSDB processes running	1 is the only sane value Also lists the vehicle's team, if one is given; see the MOOS-IvP section

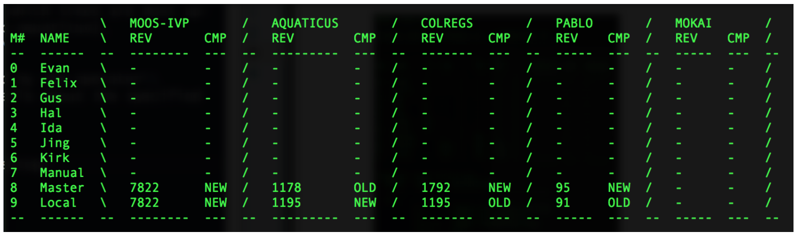
2.5.2 Network



```
              F                                    B
M#  NAME   ID     PING  SSH  USER     ADDR              PING  SSH  USER         ADDR
--  ------ --  --  ----  ---  -------  ------------ --  ----  ---  -----------  -------------------------
0   Evan   5   /   !     !    student  192.168.5.1  /   !     !    student2680  192.168.5.100
1   Felix  6   /   !     !    student  192.168.6.1  /   !     !    student2680  192.168.6.100
2   Gus    7   /   !     !    student  192.168.7.1  /   !     !    student2680  192.168.7.100
3   Hal    8   /   !     !    student  192.168.8.1  /   !     !    student2680  192.168.8.100
4   Ida    9   /   !     !    student  192.168.9.1  /   !     !    student2680  192.168.9.100
5   Jing   10  /   !     !    student  192.168.10.1 /   !     !    student2680  192.168.10.100
6   Kirk   11  /   !     !    student  192.168.11.1 /   !     !    student2680  192.168.11.100
7   Manual     /   NA    NA                         /   !     !    student      192.168.1.192
8   Master     /   NA    NA                         /   OK    OK   student2680  pablo-master.csail.mit.edu
9   Local      /   NA    NA                         /   LOC   LOC               localhost
--  ------ --  --  ----  ---  -------  ------------ --  ----  ---  -----------  -------------------------
```

Topic	Explanation	Comments
M#		See the Main view
Name		See the Main view
ID		See the Main view
F	Front Seat block	
PING	Is ADDR reachable by ping	NA indicates no front seat expected
SSH	If USER@ADDR can run a simple test command	NA indicates no front seat expected
USER	The front seat username	
ADDR	The front seat address	
B	Back Seat block	Single-computer vehicles are considered back seats
PING	Is ADDR reachable by ping	
SSH	If USER@ADDR can run a simple test command	
USER	The back seat username	
ADDR	The back seat address	

2.5.3 SVN

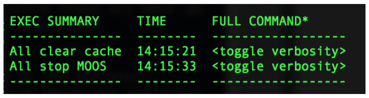


```
            \   MOOS-IVP        /   AQUATICUS       /   COLREGS        /   PABLO       /   MOKAI        /
M#  NAME    \   REV       CMP   /   REV        CMP  /   REV      CMP   /   REV    CMP  /   REV    CMP   /
--  ------  --  --------  ---   --  ---------  ---  --  -------  ---   --  -----  ---  --  -----  ---   --
0   Evan    \   -         -     /   -          -    /   -        -     /   -      -    /   -      -     /
1   Felix   \   -         -     /   -          -    /   -        -     /   -      -    /   -      -     /
2   Gus     \   -         -     /   -          -    /   -        -     /   -      -    /   -      -     /
3   Hal     \   -         -     /   -          -    /   -        -     /   -      -    /   -      -     /
4   Ida     \   -         -     /   -          -    /   -        -     /   -      -    /   -      -     /
5   Jing    \   -         -     /   -          -    /   -        -     /   -      -    /   -      -     /
6   Kirk    \   -         -     /   -          -    /   -        -     /   -      -    /   -      -     /
7   Manual  \   -         -     /   -          -    /   -        -     /   -      -    /   -      -     /
8   Master  \   7822      NEW   /   1178       OLD  /   1792     NEW   /   95     NEW  /   -      -     /
9   Local   \   7822      NEW   /   1195       NEW  /   1195     OLD   /   91     OLD  /   -      -     /
--  ------  --  --------  ---   --  ---------  ---  --  -------  ---   --  -----  ---  --  -----  ---   --
```

Topic	Explanation	Comments
M#		See the Main view
Name		See the Main view
ABC REV	Revision number of the copy of ABC	
ABC CMP	ABC tree is comparatively OLD or NEW(est)	Contacting a new machine may change who is newest

The tracked trees are `moos-ivp`, `moos-ivp-aquaticus`, `moos-ivp-colregs`, `pablo-common` and `mokai-common`. The PABLO and Mokai trees tend to not coexist, so they are special cased on the Main view such that having one but not the other will not bubble up an error.

2.5.4 History



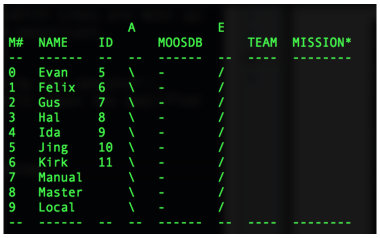
```
EXEC SUMMARY     TIME      FULL COMMAND*
---------------  --------  ------------------
All clear cache  14:15:21  <toggle verbosity>
All stop MOOS    14:15:33  <toggle verbosity>
---------------  --------  ------------------
```

Topic	Explanation	Comments
EXEC SUMMARY	Explains the command	Most recent is is displayed in the footer
TIME	Time command was dispatched	Local computer time
Full Command	Full command as sent over the wire	Often very large; toggle verbosity to read

Only the last ten commands are displayed.

Note: the astersik in the header indicates that the topic has multiple verbosity modes.

2.5.5 MOOS-IvP



```
                A            E
M#  NAME    ID      MOOSDB       TEAM  MISSION*
--  ------  --  --  ------   --  ----  --------
0   Evan    5   \   -        /
1   Felix   6   \   -        /
2   Gus     7   \   -        /
3   Hal     8   \   -        /
4   Ida     9   \   -        /
5   Jing    10  \   -        /
6   Kirk    11  \   -        /
7   Manual      \   -        /
8   Master      \   -        /
9   Local       \   -        /
--  ------  --  --  ------   --  ----  --------
```

Topic	Explanation	Comments
M#		See the Main view
Name		See the Main view
ID		See the Main view
A	Actual results block	Values here read off the target machine
MOOSDB		See the Main view
		Note that team isn't included here, unlike in the Main view
E	Expected results block	Values here are what uFleetManager would dispatch
Team	Team that the machine is on	Read from config
Mission	Launch file and args	If this is blank, startMOOS doesn't dispatch anything
		Toggle verbosity to see full path

Note: the astersik in the header indicates that the topic has multiple verbosity modes.

2.6 Commands

Many of these commands require the operator's fleet manager to be in "commanding mode"; they will be indicated by a * next to their name in this list. Some of these commands require confirmation; they will be indicated with a $ next to their name in this list.

Command	Key Feed	Description
Quit	ctrl-c	Close uFleetManager
Help	ctrl-h	Toggle help text; default is most hidden
CMD mode	ctrl-a	Toggle command mode; default is not in command
Verbose mode	V	Toggle verbose mode; default is terse
Clear	Backspace	Clear key feed
Start MOOS*	S	Start MOOS on each available machine, if possible
	s#	Start MOOS on machine #, if possible
Stop MOOS*$	K	Stop MOOS on all available machines (aka `ktm`)
	k#	Stop MOOS on machine #
Restart MOOS*$	R	Equivalent to the sequence `K S`
	r#	Equivalent to the sequence `k# s#`
Reboot Machine*$	W	Reboot all the machines (back seats)
	w#	Reboot machine #'s back seat
Shutdown Machine*$	D	Shutdown all the machines (back seats)
	d#	Shutdown machine #'s back seat
Reboot Vehicle*$	G	Reboot each of the machines' front seats, if they have them
	g#	Reboot machine #'s front seat, if it has one
Shutdown Vehicle*$	F	Shutdown each of the machines' front seats, if they have them
	f#	Shutdown machine #'s front seat, if it has one

2.7 Config files

Config files use standard `.moos` file syntax. The minimal config file looks like this

```
ProcessConfig = uFleetManager

{
 machines = alpha bravo charlie

}
```

and this would direct the Fleet Manager to watch three known vehicles; `Alpha`, `Bravo`, and `Charlie`, without specifying their mission or team.

To specify their team, add team variables like so:

```
ProcessConfig = uFleetManager

{

 machines = alpha bravo charlie

 red = alpha

 blue = bravo delta

}
```

This will result in `Alpha` and `Bravo` being assigned teams (which will show up in the `main` and `MOOS` views), `Charlie` will not have a team, and `Delta`'s team would be ignored because `Delta` isn't called out in `machines`.

To specify a mission directory, add the full path;

```
 all_mission_dir = ~/some/fully/qualified/path/
```

To specify vehicle-specific arguments for vehicle `foobar`, add a variable with its name;

```
 foobar = launch:launch.sh, dir:some/other/path, dir_rel:true, args:-s
```

Which would instruct `foobar` to use the `launch.sh` launch file when given the command to launch MOOS-IvP by the Fleet Manager, and to look for it at `~/some/fully/qualified/path/some/other/path`, and pass it the argument `-s`. If `dir_rel` was given `false`, then the Fleet Manager would go instead to `some/other/path` and look for `launch.sh` there.

As before, specifying the arguments for `foobar` will be ignored if `foobar` isn't in the `machines` list.

Comments are `//`.

The list of available machines is aqua1, aqua2, aqua3, evan, felix, gus, hal, ida, jing, kirk, master, and manual. To add more, edit `Configuration.cpp`

3 Modifying the Fleet Manager

3.1 Adding Views

There are four places in `ui.cpp` that need to be modified to add a view; the help text, the table formatting, the navigation character handlers, and the view render block.

Help Text Find the block in `UI::setTableFormats()` of additions to `m_help["nav"]`; the syntax is a struct of three strings;

```
{view name, navigation character, help text description}
```

Table Formats Find the blocks in `UI::setTableFormats()` like

```
foo.push_back("BLAH")

foo.push_back("BLAH BLAH")

m_headers["foobar"].push_back(foo)
```

The map `m_headers` stores the headers for each view. A header is a vector of vectors of strings; the outer vector stores rows to feed to ACTables, and the inner vector stores the strings to put in each column of the table. Sections that have multiple header rows should be specified as

```
foo1.push_back("BLAH"); foo2.push_back("DUH")

foo1.push_back("BLAH BLAH"); foo2.push_back("DUH DUH")

m_headers["foobar"].push_back(foo1);

m_headers["foobar"].push_back(foo2);
```

the first block would result in a table formatted like

BLAH	BLAH BLAH
...	...

while the second block would result in a table formatted like

BLAH	BLAH BLAH
DUH	DUH DUH
...	...

Usually one or two lines is sufficient; the first line to delineate sections and the second line for column headers. Add your own section, consistent with what you put in the Help Text section. You will revisit this in Adding Topics to a View.

Character Handlers Find the block in `UI::actOnKeyPress()` with sequences like

```
else if (m_key_feed=="M") {

 m_view = "MOOS";
 command_match = true;

}
```

and add you own, consistent with the information you put in the Help Text section

View Render Find the block in `UI::printWindow()` that looks like

```
if (m_view=="FOO") {

 view_table << something

}

else if (m_view=="BAR") {

 view_table << something else

}

...
```

and add a similar block checking for your new view. See the next section for how to fill out that block.

3.2 Adding Topics to a View

There are two places in `ui.cpp` that need to be modified to add a column to a view; the table formatting block and the view rendering block.

Table Formats Find your block in `UI::setTableFormats()`, the same as your Table Formats block from Adding Views. Add a string to all the inner vectors. Disallowed[1] strings include ”\n” and ”|”, and allowable strings include ””, ”\”, ”/”, and ”#”.

View Render Find your block in `UI::printWindow()`, the same as your View Render block from Adding Views. The `n`th line such as `view_table << something` will fill the `n`th column of the table as ordered in Table Formats.

3.3 Adding Commands

The interface for the UI to call vehicle commands, to get information or to take action, is public `ManagedMoosMachine` methods.

Commands are fired off into the void, with a file to write results back to. These files are opened and read synchronously with the local machine, with a small proability[2] of reading partially written data[3]. This architecture approximates threading[4], but does not require maintainers to understand threading per se.

[1] Used by ACTables for formatting.

[2] Determined by the duty cycle of file IO

[3] uFleetManager's networking layer is written such that in that case, the message ID is the last thing written, and only once it is complete will the app do anything with that data. In formal terms, this satisfies only the Consistency pillar of CAP. If the user clears the cache agggressively, it also weakly satisfies Partition Tolerance.

[4] This architecture was selected in keeping with Dr. Benjamin's standing instructions that any user with basic

3.3.1 Dispatching

At the high level, the fleet manager is a big wrapper for sending commands over `ssh`;

```
ssh ADDR "remote_cmd"
```

The complexity in sending commands is in letting go of it so the app can return to its thread of execution. The normal way to execute commands from `C++` is with the `system_call()` function[5], which is fine for local, synchronous commands. However, that naive approach is not sufficient to have many asynchronous commands in flight at the same time. The solution involves `nohup` (no hangup) and `&`, and is implemented in the `_dispatch()` function in `system_call.cpp`[6].

The robust interface from `system_call.cpp` is two functions, `system_call_dispatch_pipe()` and `system_call_dispatch_return()`. Both of them dispatch commands and capture an output from the script via `ssh` and write it to a named mailbox; `_pipe()` captures from `stdout`, where `_return()` captures the script's return value.

When you're adding a new public method to `ManagedMoosMachine`, it will be essentially a wrapper around either `system_call_dispatch_pipe()` or `system_call_dispatch_return()`. There are two common ways to do so; standard PAVLAB commands, and one-off commands. All else being equal, standard commands are preferable.

3.3.2 Standard PAVLAB Commands

The standard PAVLAB way of interfacing with lab machines outside of MOOS-IvP itself is via a *machine*-common directory. Currently we are maintaining two lab `svn` repos, `pablo-common` and `mokai-common`, and they each contain a directory called FleetManagerScripts with the lab's standard scripts. Commands have a simple naming scheme: "pav_*action*_*object*.sh". Some examples are

```
pav_test_ssh.sh

pav_get_svn_rev_moos.sh

pav_up_svn_moos.sh

pav_reboot_computer.sh
```

The sole exception is `pav_not_implemented.sh`, the placeholder implementation, which lacks a verb (such as "is").

All machines should have their relevant kind of *machine*-common tree, but they will have the Anonymous Read Only version, *machine*-common-aro[7].

C++ and Bash experience should be able to understand any code in the lab. The PAVLAB considers threading a non-basic feature.

[5] system_call() has known security issues, be very careful using it unless you're absolutely sure you know the pedigree of the scripts you're calling with it.

[6] Note, I left some vestigal code about timeouts - I was leaking background processes, and attempting to solve that by sending out the scripts with a kill switch on a timer. Instead, the eventual solution was to use message indices and only send out a new request once the old one returned, I just haven't had time to clean up that bit of code.

[7] This allows anyone to call update and to use the scripts, but not to push changes. To push changes, talk to Dr. Benjamin about getting access to the *machine*-common repos

There is a special helper function in ManagedMoosMachine to streamline calling those standard PAVLAB commands, _dispatchPavCmd(). There are several good examples of its usage in the ManagedMoosMachine class.

3.3.3 Special Commands

Some commands do not lend themselves to the common and standardized system. For example, ping makes no sense to be hosted remotely. Rebooting computers can be configured to run without a password on some operating systems (e.g. Raspian) but it's not clear on others (e.g. Ubuntu) so one-off versions are sometimes needed.

In that case, compose the script in code and use the appropriate dispatcher (system_call_dispatch_pipe() or system_call_dispatch_return()) to run it.

One interesting caveat is that unlike normal scripts, where instructions are separated by semicolons, in these scripts the instructions must be separated by newlines.

3.3.4 Reading Mail

Dispatched commands will, once they conclude, yield a result to a mailbox file. Mailbox files are files in the directory /tmp/MOOSMAIL. The usual naming is /tmp/MOOSMAIL/*Machine_commandName*.mailbox. For ease of use, use the ManagedMoosMachine helper function serviceMailboxName().

Once that result is put in the mailbox, you'll want to read it. At its core, we're just reading lines out of the mailbox and parsing them. By checking for message indices, this step also serves as a caching and synchronizing step.

Consider a new ManagedMoosMachine public method, checkFooServiceMail(); its implementation might look something like this:

```
vector<string> mail_list = readServiceMailbox(m_mail["fooService"].mailbox);

index_t index = grabIndex(mail_list);

if (receiveUpdate(m_mail["fooService"].cache, index)) {

 // do parsing here...

 m_mail["fooService"].cache.data = /* a result string */

}

return(get_data_and_staleness(m_mail["fooService"].cache));
```

When returning a status, consider looking in the Status namespace in Constants.h. These statuses are shared throughout the app, allowing them to be reasoned over in the UI.

3.3.5 Required Variables and Caching

The infrastucture of the ManagedMoosMachine is in the m_mail map; it contains the cache (of type StampedData, refer to Utils.h) and the mailbox file name.

To add a new required variable, find the ManagedMoosMachine constructor, all you have to do is add a line like

```
m_mail["foobar"].mailbox = serviceMailboxName("fooBar");
```

near the others that follow that pattern.

Bracket lookup adds foobar to the m_mail map, and then sets the name of the mailbox it uses for script writebacks. Below, it loops through all m_mail entries (which now includes foobar) and adds a blank StampedData struct. After rebuilding and running, if you look in the mailbox directory (by default, /tmp/MOOSMAIL), you should now see an Alpha_fooBar.mailbox, a Bravo_fooBar.mailbox, etc. Their contents will be whatever you asked to be written back, plus an index at the end:

```
MOOS_MANAGER_MESSAGE_INDEX:N
```

which is part of the networking solution described in Adding Commands and Dispatching. Once you call grabIndex(), that line will be stripped out and you will only be presented with the payload.

3.4 Miscellaneous

1. To add new vehicles, look to Configuration.cpp

3.4.1 Tips and Tricks

1. If you're getting strange status results, always check the actual contents of the mailbox:

   ```
   cat /tmp/MOOSMAIL/that.mailbox
   ```

   or if you want to get really fancy

   ```
   while true; do cat /tmp/MOOSMAIL/that.mailbox; sleep 1; clear; done
   ```

2. If you're trying to debug script output that's going to stdout once and then disappearing, try adding a sleep(seconds); to the main render loop in ui.cpp. In fact, there's one already there commented out for just that eventuality.

4 Configuring Machines to work with the Fleet Manager

4.1 SSH Keys

In order to script commands over ssh, the target machine needs your ssh public key to allow you to log on with no password.

The *machine*-common repos have scripts that should update the target machine's ~/.ssh/authorized_keys with your public key if it's added into the *machine*-common/*machine*_authorized_keys file.

For machines that don't have a *machine*-common repo, or if you can't get a hold of someone with editing privileges, you can add it in manually.

4.2 Shell Startup and Sources

In order to have the PAVLAB commands available both for interactive shells that you start when you log in and the non-interactive shells that uFleetManager starts, you need to source it in the right place in your ~/.profile or ~/.bashrc files. Many of them have blocks like:

```
# If not running interactively, don't do anything

case $- in

 *i*) ;;

 *) return;;

esac
```

Your block must be before anything like that. Your block should look much like this:

```
# Source the svn-controlled bashrc or cshrc

if [ $SHELL = "/bin/bash" ]; then

 .  "$HOME/pablo-common-aro/dot_bashrc"

elif [ $SHELL = "/bin/csh" ]; then

 .  "$HOME/pablo-common-aro/dot_cshrc"

fi
```

Substituting pablo-common, mokai-common, or mokai-common-aro as appropriate.

4.3 Software

All target machines should have a copy, ARO or otherwise, of at least one *machine*-common tree like pablo-common or mokai-common.

Target machines should also have MOOS-IvP and any other MOOS-IvP trees (e.g. Aquaticus and Colregs) necessary for the missions.

4.4 Permissions

A few select commands require special permissions, namely shutting down and rebooting computers. Each operating system has different amounts that you can deprotect those two commands, and we have explored those for Raspbian and Ubuntu:

Raspbian Run

```
sudo visudo -f /etc/sudoers.d/shutdown
```

and add the lines

```
student2680 host = (ALL) NOPASSWD: /sbin/shutdown now

student2680 host = (ALL) NOPASSWD: /sbin/reboot
```

then run

```
sudoedit /etc/polkit-1/localauthority/50-local.d/reboot.pkla
```

and add the lines

```
[Shutdown]

Identity=unix-user:student2680

Action=org.freedesktop.login1.power-off

ResultAny=yes


[Reboot]

Identity=unix-user:student2680

Action=org.freedesktop.login1.reboot

ResultAny=yes
```

and now the two commands `shutdown now` and `reboot` are completely unprotected by `sudo`

Ubuntu Run

```
sudo visudo -f /etc/sudoers.d/shutdown
```

and then add the lines

```
student ALL=NOPASSWD: /sbin/shutdown now

student ALL=NOPASSWD: /sbin/reboot
```

and now the two commands `shutdown now` and `reboot` can be run with `sudo`, so `sudo shutdown now` and `sudo reboot`, but will not require a password. When run over `ssh`, it still requires `sudo` in the command and the `-t` argument to `ssh`, but will not require a password.

5 Debugging

Symptom	Likely Issue	Resolution
Semicolons in command (see History)	Needs to be \n	Replace in relevant ManagedMoosMachine dispatcher function.
App responds slowly	Resource starved	Check CPU usage Check process count Try closing Google Chrome or other resource-hogging programs Restart uFleetManager
No machines seen	On wrong network	Switch to kayak-local
	Wifi is down	Check that the Bullet is on and working
Password requests in `stdout`	Target needs ssh keys	Add your public key to ∼`/.ssh/authorized_keys` on the target machine