Payload UUV Autonomy
(3 Architecture Principles)

Architecture Principle #1
Payload Autonomy
Decouple the Procurement of Hardware and Software
Payload UUV Autonomy
(3 Architecture Principles)

Architecture Principle #2
Autonomy System Middleware
De-couple Software Procurements
Sensing, Autonomy, Simulation, Comms...

MOOS Middleware
MOOS Applications

Payload Computer
Main Vehicle Computer

Payload Computer

Sensing
Autonomy
Comms
Contact Management
MOOS Applications

Simulation
MOOS Application

Payan UUV Autonomy
(3 Architecture Principles)

Behavior-Based
Modular HELM
MOOS Overview

- MOOS is a kind of Robot Middleware
- Developed by Paul Newman, as an MIT post-doc and now Oxford Professor
- Initial development 2000-2003 on Bluefin Odyssey II UUV owned by MIT
MOOS in the Oxford Autonomous Vehicle Project

- Three Autonomous Vehicles:
  - 2 Nissan EV’s, one off road 4x4
  - 20Hz stereo cameras 32Mb/s throughput
- 7 Lasers at 50Hz
- 5 CCD omni directional camera
- 50Hz realtime control of 2e3 kg vehicle

Nested Repositories

- MOOS, from the Mobile Robotics Group at Oxford
- MOOS-IvP, from the Laboratory for Autonomous Marine Sensing Systems at MIT
- 3rd Party (Your) modules.
MOOS Does Two Main Things

1. It enables distinct applications to communicate
2. It enables users to control the frequency of each application’s action loop

The Beauty of Separate Processes

On Unix based systems, each process:
- Has a unique Process ID (PID)
- Uses a chunk of computer memory separate from all other processes

Advantages:
- A crash in one process will not affect another process
- The OS automatically distributes processes over system CPU cores
MOOSDB is a Process for Communication

- It has its own PID and memory space like any other process
- It maintains a mapping for Variable Names → Values

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FRUIT</td>
<td>apples</td>
</tr>
<tr>
<td>ANGLE</td>
<td>135</td>
</tr>
<tr>
<td>SPEED</td>
<td>2.8</td>
</tr>
<tr>
<td>NAME</td>
<td>alpha</td>
</tr>
<tr>
<td>WIDTH</td>
<td>86</td>
</tr>
<tr>
<td>HOURS</td>
<td>23</td>
</tr>
</tbody>
</table>

Only the most recent value is retained

MOOS Apps Subscribe to the MOOSDB

- An App may register (subscribe for) for any variable
- An App may register any time, but typically during startup

When an App first connects, it gets mail for each registered variable. (if the variable has ever been written to)
MOOS Apps Publish to the MOOSDB

- An App may publish to the MOOSDB any time
- No prior arrangement required

Note: Subscribers will get all postings – each as a new piece of mail.

MOOSDB

<table>
<thead>
<tr>
<th>FRUIT</th>
<th>apples</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANGLE</td>
<td>22</td>
</tr>
<tr>
<td>SPEED</td>
<td>2.8</td>
</tr>
<tr>
<td>NAME</td>
<td>beta</td>
</tr>
<tr>
<td>WIDTH</td>
<td>86</td>
</tr>
<tr>
<td>HOURS</td>
<td>8.4</td>
</tr>
</tbody>
</table>

Application Process A

ANGLE = 45
NAME = beta

Application Process B

ANGLE = 22
SPEED = 2.9
HOURS = 8.4

A MOOS Community

- A MOOS community is comprised of one MOOSDB and all connected Apps
- MOOS is described as having a star topology.

A community also has a unique
- name
- IP address, Port number
A MOOS Community Per Robot

- Typically one community per vehicle/robot
- Sometimes multiple computers on one vehicle, each with a community

Community 1

Community 2

- Inter-community communications addressed later

Example: The Alpha Mission

```bash
$ cd moos-lvp/lvp/missions/s1_alpha
$ ./launch.sh 10
```
Three Architectures

MOOS Overview

MOOS Messages

Launching Missions

Scoping MOOS

Poking MOOS

Data Logging

Michael Benjamin, Henrik Schmidt, Spring 2018

MIT Dept of Mechanical Engineering

### Alpha Mission - Modules

$ cd moos-ivp/ivp/missions/s1_alpha
$ ./launch.sh 10
Payload UUV Autonomy
(3 Architecture Principles)

Main Vehicle Computer

Payload Computer

MOOS

You

MOOSDB

MOOSApp Superclass

Dedicated Application

A few important applications come with MOOS:
- pAntler
- pLogger
- pShare

Three Architectures
MOOS Overview
MOOS Messages
Launching Missions
Scoping MOOS
Poking MOOS
Data Logging

Michael Benjamin, Henrik Schmidt, Spring 2018
MIT Dept of Mechanical Engineering
An autonomy system has components with different capabilities, and distribution access.
- Publicly accessible modules providing infrastructure, basic capabilities
- Restricted-access modules for developers of a particular project.

Autonomy System = Infrastructure + Modules

Core Infrastructure. Open Source.

Project specific add-on modules. Non Open Source.
MOOS Messages

- Two primary message components: VARIABLE and VALUE
- Two primary message types: STRING and DOUBLE

### MOOSDB

<table>
<thead>
<tr>
<th>Name</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>FRUIT</td>
<td>apples</td>
</tr>
<tr>
<td>ANGLE</td>
<td>135</td>
</tr>
<tr>
<td>SPEED</td>
<td>2.8</td>
</tr>
<tr>
<td>NAME</td>
<td>alpha</td>
</tr>
<tr>
<td>WIDTH</td>
<td>86</td>
</tr>
<tr>
<td>HOURS</td>
<td>23</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>StringVal</th>
<th>DoubleVal</th>
</tr>
</thead>
<tbody>
<tr>
<td>StringVal</td>
<td>Data in human-readable string format, or raw binary data</td>
<td></td>
</tr>
<tr>
<td>DoubleVal</td>
<td>Numeric double float data</td>
<td></td>
</tr>
</tbody>
</table>

### MOOS Message Examples

<table>
<thead>
<tr>
<th>MOOSDB</th>
</tr>
</thead>
<tbody>
<tr>
<td>FRUIT apples</td>
</tr>
<tr>
<td>ANGLE 135</td>
</tr>
<tr>
<td>SPEED 2.8</td>
</tr>
<tr>
<td>NAME alpha</td>
</tr>
<tr>
<td>WIDTH 86</td>
</tr>
<tr>
<td>HOURS 23</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>StringVal</th>
<th>DoubleVal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>StringVal</td>
<td>DoubleVal</td>
</tr>
<tr>
<td>WIDTH</td>
<td>string</td>
<td>86</td>
</tr>
</tbody>
</table>
MOOS Messages

Each MOOS Message contains additional useful information:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StringVal</td>
<td>Data in human-readable string format, or raw binary data</td>
</tr>
<tr>
<td>DoubleVal</td>
<td>Numeric double float data</td>
</tr>
<tr>
<td>DataType</td>
<td>Type of data (STRING or DOUBLE or BINARY)</td>
</tr>
<tr>
<td>Source</td>
<td>Name of client that sent this data to the MOOSDB</td>
</tr>
<tr>
<td>SourceAux</td>
<td>Optional additional information about the source client</td>
</tr>
<tr>
<td>Time</td>
<td>Time at which the data was written</td>
</tr>
<tr>
<td>Community</td>
<td>The community to which the source process belongs</td>
</tr>
</tbody>
</table>

Posting MOOS Messages

Inside your MOOS application you may post a message with simple line in C++:

```
Notify("FRUIT", "apples");
```

MOOS will automatically fill in the additional fields:
Reading MOOS Messages

- MOOS Apps read messages inside a mail-handling function
- This function is defined in the MOOSApp superclass for all MOOS Apps

The Flow of Control for all MOOS Apps

A Mail Handling Example

- An example OnNewMail implementation:

```cpp
bool MyApp::OnNewMail(MOOSMSG_LIST &NewMail)
{
  MOOSMSG_LIST::iterator p;
  for(p=NewMail.begin(); p!=NewMail.end(); p++) {
    CMOOSMsg &msg = *p;
    string key = msg.GetKey();
    if(key == "WIDTH")
      updateWidth(msg.getDouble());
  }
  return(true);
}
```
Handling a MOOS Message

Other useful functions defined on a MOOS Message:

```cpp
MOOSMsg msg;
string moos_var = msg.GetKey();         // the MOOS variable name
bool is_double = msg.IsDouble();        // true if message content double
bool is_string = msg.IsString();         // true if message content string
double timestamp = msg.GetTime();       // timestamp when message posted
string str_val = msg.GetString();       // the message string content
string dbl_val = msg.GetDouble();       // the message double content
string source = msg.GetSource();        // who (which app) posted message
string src_aux = msg.GetSourceAux();    // further source information
string community = msg.GetCommunity();  // MOOS community who posted
```
Launching MOOS (Bare Bones)

The MOOSDB may be launched from the command line:

```
$ MOOSDB
```

- The new MOOSDB process is the beginning of a MOOS community
- Recall a community has an IP Address, Port Number, Community Name

Terminal output:

```
--------------------------------- MOOSDB V10 --------------------------------
Hosting community   "#1"  
Name look up is      off     
Asynchronous support is on   
Connect to this server on port 9000   
--------------------------------- MOOSDB V10 --------------------------------

network performance data published on localhost:9020   
listen with "nc -u -lk 9020"
```

Launching MOOS (with Mission File)

- The IP Address, Port Number and Community Name may be provided in a mission file.
- The mission file is a command line argument:

```
$ MOOSDB mission.moos
```

```
--------------------------------- MOOSDB V10 --------------------------------
Hosting community   "alpha"  
Name look up is      off     
Asynchronous support is on   
Connect to this server on port 9205   
--------------------------------- MOOSDB V10 --------------------------------

network performance data published on localhost:9020   
listen with "nc -u -lk 9020"
```
Launching MOOS and Mission Configuration

- A mission file may also hold configuration parameters for MOOS apps
- Each application has a dedicated configuration block.

```plaintext
mission.moos

Global parameters

ProcessConfig = alpha
{
    alpha parameters
}

ProcessConfig = bravo
{
    alpha parameters
}
```

MOOS Mission Configuration

Mission configuration is through a single “mission file”, with a .moos extension. Each application has a dedicated configuration block.

```plaintext
mission.moos

Global parameters

ProcessConfig = alpha
{
    alpha parameters
}

ProcessConfig = bravo
{
    alpha parameters
}
```

“Global parameters” are accessible to all MOOS applications. They include things like:
- MOOSDB server IP address and port number.
- Local datum (0,0) in lat/lon coordinates.
- Name of the MOOS community.
MOOS Mission Configuration

Mission configuration is through a single “mission file”, with a .moos extension. Each application has a dedicated configuration block.

```moos
mission.moos

Global parameters
ProcessConfig = alpha
{ alpha parameters }
ProcessConfig = bravo
{ alpha parameters }
```

“Application parameters”
Accessible only to a particular application.

Application authors implement the handling of parameters upon application startup.

The MOOSApp superclass has a function called OnStartUp() where configuration parameters are handled.

Application authors have access to each line in the application’s configuration block to handle as they see fit.

Scoping MOOS
Scoping MOOS

Scoping the MOOSDB means examining:
- Current values of variables known to the MOOSDB
- Which processes made the most recent post
- When it was posted
- The community of the application making the post.

### MOOSDB

<table>
<thead>
<tr>
<th>VarName</th>
<th>Source</th>
<th>Community</th>
<th>Time</th>
<th>VarValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>FRUIT</td>
<td>pFruit</td>
<td>alpha</td>
<td>143.21</td>
<td>apples</td>
</tr>
<tr>
<td>ANGLE</td>
<td>uMeasure</td>
<td>alpha</td>
<td>1873.24</td>
<td>135</td>
</tr>
<tr>
<td>SPEED</td>
<td>uMeasure</td>
<td>alpha</td>
<td>62.11</td>
<td>2.8</td>
</tr>
<tr>
<td>NAME</td>
<td>pIdentity</td>
<td>gamma</td>
<td>3.91</td>
<td>alpha</td>
</tr>
<tr>
<td>WIDTH</td>
<td>uMeasure</td>
<td>alpha</td>
<td>1873.24</td>
<td>86</td>
</tr>
<tr>
<td>HOURS</td>
<td>uMeasure</td>
<td>alpha</td>
<td>1873.25</td>
<td>23</td>
</tr>
</tbody>
</table>
A Simple Single Scope in pMarineViewer

The pMarineViewer app has a single simple scoping field.

Changing the Scope Variable in pMarineViewer

The scope variable may be changed:
Hit SHIFT-'A'
Enter a new variable.
The uXMS Scope List

uXMS is a simple scoping utility launched from the command line

$ uXMS mission.moos --all

- To scope on a MOOSDB, uXMS must connect to the MOOSDB.
- Where is the server?
- It could be anywhere on the internet.
- Exactly where? This is determined by the IP address and the port number, for the MOOSDB server.
The uXMS Scope List

uXMS is a simple scoping utility launched from the command line

```
$ uXMS mission.moos --all
```

- To scope on a MOOSDB, uXMS must connect to the MOOSDB.
- Where is the server?
- It could be anywhere on the internet.
- Exactly where? This is determined by the IP address and the port number, for the MOOSDB server.

The same information may also be passed on the command line as arguments to uXMS

```
$ uXMS --all --serverhost=localhost --serverport=9005
```
The uXMS Scope List

uXMS is a simple scoping utility launched from the command line

\$ uXMS mission.moos --all

By default, the screen will refresh whenever one variable value changes

```
VarName   (S)ource   (T)ime   (C)ommunity VarValue
--------- -------------- ------- --------------- ---------
DB_CLIENTS MOOSDB_alpha 106.2 alpha  "uXMS,DBWebServer,"
DB_TIME    MOOSDB_alpha  107.2 alpha  1325701208.08963
DB_UPTIME  MOOSDB_alpha  107.2 alpha  107.20791
FRUIT      pFruit        143.21 alpha  "apples"
ANGLE      uMeasure      107.2 alpha  135
SPEED      uMeasure      107.2 alpha  2.8
NAME       pIdentity     3.91   gamma  "alpha"
WIDTH      uMeasure      1873.24 alpha  86
HOURS      uMeasure      1873.25 alpha  23
```

-- displaying all variables --

All scoped variables

Name of the app that last published the variable
The uXMS Scope List

uXMS is a simple scoping utility launched from the command line

```
$ uXMS mission.moos --all
```

By default, the screen will refresh whenever one variable value changes

```
VarName     | Source | Time | Community   | VarValue
-----------|--------|------|-------------|---------
DB_CLIENTS | MOOSDB_alpha | 106.2 | alpha        | "uXMS,DBWebServer,"
DB_TIME    | MOOSDB_alpha | 107.2 | alpha        | 1325701208.08963
DB_UPTIME  | MOOSDB_alpha | 107.2 | alpha        | 107.20791
FRUIT      | pFruit    | 143.21 | alpha        | "apples"
ANGLE      | uMeasure  | 107.2 | alpha        | 135
SPEED      | uMeasure  | 107.2 | alpha        | 2.8
NAME       | pIdentity | 3.91  | gamma        | "alpha"
WIDTH      | uMeasure  | 1873.24 | alpha    | 86
HOURS      | uMeasure  | 1873.25 | alpha    | 23

-- displaying all variables --
```

The community of the app that made the last publication

```
$ uXMS mission.moos --all
```

By default, the screen will refresh whenever one variable value changes

```
VarName     | Source | Time | Community   | VarValue
-----------|--------|------|-------------|---------
DB_CLIENTS | MOOSDB_alpha | 106.2 | alpha        | "uXMS,DBWebServer,"
DB_TIME    | MOOSDB_alpha | 107.2 | alpha        | 1325701208.08963
DB_UPTIME  | MOOSDB_alpha | 107.2 | alpha        | 107.20791
FRUIT      | pFruit    | 143.21 | alpha        | "apples"
ANGLE      | uMeasure  | 107.2 | alpha        | 135
SPEED      | uMeasure  | 107.2 | alpha        | 2.8
NAME       | pIdentity | 3.91  | gamma        | "alpha"
WIDTH      | uMeasure  | 1873.24 | alpha    | 86
HOURS      | uMeasure  | 1873.25 | alpha    | 23

-- displaying all variables --
```

The community of the app that made the last publication
The uXMS Scope List

uXMS is a simple scoping utility launched from the command line

```
$ uXMS mission.moos --all
```

By default, the screen will refresh whenever one variable value changes

<table>
<thead>
<tr>
<th>VarName</th>
<th>Source</th>
<th>Time</th>
<th>Community</th>
<th>VarValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB_CLIENTS</td>
<td>MOOSDB_alpha</td>
<td>106.2</td>
<td>alpha</td>
<td>&quot;uXMS, DBWebServer,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1325701208.08963</td>
</tr>
<tr>
<td>DB_TIME</td>
<td>MOOSDB_alpha</td>
<td>107.2</td>
<td>alpha</td>
<td>107.20791</td>
</tr>
<tr>
<td>DB_UPTIME</td>
<td>MOOSDB_alpha</td>
<td>107.2</td>
<td>alpha</td>
<td>&quot;apples&quot;</td>
</tr>
<tr>
<td>FRUIT</td>
<td>pFruit</td>
<td>143.21</td>
<td>alpha</td>
<td>135</td>
</tr>
<tr>
<td>ANGLE</td>
<td>uMeasure</td>
<td>107.2</td>
<td>alpha</td>
<td>2.8</td>
</tr>
<tr>
<td>SPEED</td>
<td>uMeasure</td>
<td>107.2</td>
<td>alpha</td>
<td>&quot;alpha&quot;</td>
</tr>
<tr>
<td>NAME</td>
<td>pIdentity</td>
<td>3.91</td>
<td>gamma</td>
<td>86</td>
</tr>
<tr>
<td>WIDTH</td>
<td>uMeasure</td>
<td>1873.24</td>
<td>alpha</td>
<td>23</td>
</tr>
<tr>
<td>HOURS</td>
<td>uMeasure</td>
<td>1873.25</td>
<td>alpha</td>
<td></td>
</tr>
</tbody>
</table>

-- displaying all variables --

The value of the last publication

Scoping on the Alpha Example Mission with uXMS

Launch the mission

```
$ cd moos-ivp/ivp/missions/s1_alpha
$ pAntler alpha.moos
```

At the start of sits motionless at the start position at point (0, -20) in local coordinates.
Scoping on the Alpha Example Mission with uXMS

Launch the mission

```
$ cd moos-ivp/ivp/missions/s1_alpha
$ pAntler alpha.moos
```

In a separate terminal window, launch uXMS with the following variables:

```
$ uXMS alpha.moos \    
  NAV_X NAV_Y \    
  NAV_SPEED \    
  NAV_HEADING \    
  DEPLOY \    
  IVPHELM_STATE \    
  MOOS_MANUAL_OVERRIDE
```

Launch the mission. Hit the DEPLOY button.
Scoping on the Alpha Example Mission with uXMS

Launch the mission

```
$ cd moos-ivp/ivp/missions/s1_alpha
$ pAntler alpha.moos
```

Launch the scope

```
$ uXMS alpha.moos. NAV_X NAV_Y NAV_SPEED NAV_HEADING
   DEPLOY IVPHELM_STATE MOOS_MANUAL_OVERRIDE
```

The uXMS Utility Refresh Mode Indicator

The uXMS refresh mode is indicated in the top right-hand corner of each report:
The uXMS Utility: The “History” Content Mode

$ uXMS mission.moos --history=DESIRED_HEADING

Successive duplicate entries are condensed into a single line with the number of duplicates indicated in parentheses.

Setting the Scope List by App Name

uXMS can be launched to scope only on variables from a given App:

$ uXMS mission.moos --src=uMeasure

<table>
<thead>
<tr>
<th>VarName</th>
<th>(S)ource</th>
<th>(T)ime</th>
<th>(C)ommunity</th>
<th>VarValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANGLE</td>
<td>uMeasure</td>
<td>107.2</td>
<td>alpha</td>
<td>135</td>
</tr>
<tr>
<td>SPEED</td>
<td>uMeasure</td>
<td>107.2</td>
<td>alpha</td>
<td>2.8</td>
</tr>
<tr>
<td>WIDTH</td>
<td>uMeasure</td>
<td>1873.24</td>
<td>alpha</td>
<td>86</td>
</tr>
<tr>
<td>HOURS</td>
<td>uMeasure</td>
<td>1873.25</td>
<td>alpha</td>
<td>23</td>
</tr>
</tbody>
</table>
Scoping LOCALLY

- Typically a scope is run on the same machine as the rest of the MOOS Community.

Scoping REMOTELY

- A scope may also connect to a remote machine
- Need to specify IP Address, Port Number:

```bash
$ uXMS mission.moos --serverhost=18.231.8.45 --serverport=9200
```
Poking MOOS

**Poking** the MOOSDB:
- A write to the MOOSDB
- Implies that it is outside a typical application write to the MOOSDB

<table>
<thead>
<tr>
<th>MOOSDB</th>
</tr>
</thead>
<tbody>
<tr>
<td>FRUIT</td>
</tr>
<tr>
<td>ANGLE</td>
</tr>
<tr>
<td>SPEED</td>
</tr>
<tr>
<td>NAME</td>
</tr>
<tr>
<td>WIDTH</td>
</tr>
<tr>
<td>HOURS</td>
</tr>
</tbody>
</table>
Changing a Variable Value with a MOOS Poke

• A poke may simply alter the variable value

Before:

MOOSDB

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FRUIT</td>
<td>apples</td>
</tr>
<tr>
<td>ANGLE</td>
<td>135</td>
</tr>
<tr>
<td>SPEED</td>
<td>2.8</td>
</tr>
<tr>
<td>NAME</td>
<td>alpha</td>
</tr>
<tr>
<td>WIDTH</td>
<td>86</td>
</tr>
<tr>
<td>HOURS</td>
<td>23</td>
</tr>
</tbody>
</table>

Poke:

NAME = “bravo”

After:

MOOSDB

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FRUIT</td>
<td>apples</td>
</tr>
<tr>
<td>ANGLE</td>
<td>135</td>
</tr>
<tr>
<td>SPEED</td>
<td>2.8</td>
</tr>
<tr>
<td>NAME</td>
<td>bravo</td>
</tr>
<tr>
<td>WIDTH</td>
<td>86</td>
</tr>
<tr>
<td>HOURS</td>
<td>23</td>
</tr>
</tbody>
</table>

Publishing a New Variable with a MOOS Poke

• A poke may write to a new MOOS variable

Before:

MOOSDB

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FRUIT</td>
<td>apples</td>
</tr>
<tr>
<td>ANGLE</td>
<td>135</td>
</tr>
<tr>
<td>SPEED</td>
<td>2.8</td>
</tr>
<tr>
<td>NAME</td>
<td>alpha</td>
</tr>
<tr>
<td>WIDTH</td>
<td>86</td>
</tr>
<tr>
<td>HOURS</td>
<td>23</td>
</tr>
</tbody>
</table>

Poke:

BAND = “Beatles”

After:

MOOSDB

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FRUIT</td>
<td>apples</td>
</tr>
<tr>
<td>ANGLE</td>
<td>135</td>
</tr>
<tr>
<td>SPEED</td>
<td>2.8</td>
</tr>
<tr>
<td>NAME</td>
<td>alpha</td>
</tr>
<tr>
<td>WIDTH</td>
<td>86</td>
</tr>
<tr>
<td>HOURS</td>
<td>23</td>
</tr>
<tr>
<td>BAND</td>
<td>beatles</td>
</tr>
</tbody>
</table>
A Poke May Not Change an Existing Variable Type

- Once a variable is of type string – it is always a string
- Once a variable is of type double – it is always a double
- Subsequent pokes are ignored

### Before

<table>
<thead>
<tr>
<th>MOOSDB</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>FRUIT</td>
<td>apples</td>
</tr>
<tr>
<td>ANGLE</td>
<td>135</td>
</tr>
<tr>
<td>SPEED</td>
<td>2.8</td>
</tr>
<tr>
<td>NAME</td>
<td>alpha</td>
</tr>
<tr>
<td>WIDTH</td>
<td>86</td>
</tr>
<tr>
<td>HOURS</td>
<td>23</td>
</tr>
</tbody>
</table>

**Poke**

WIDTH = "thin"

<table>
<thead>
<tr>
<th>MOOSDB</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>FRUIT</td>
<td>apples</td>
</tr>
<tr>
<td>ANGLE</td>
<td>135</td>
</tr>
<tr>
<td>SPEED</td>
<td>2.8</td>
</tr>
<tr>
<td>NAME</td>
<td>bravo</td>
</tr>
<tr>
<td>WIDTH</td>
<td>86</td>
</tr>
<tr>
<td>HOURS</td>
<td>23</td>
</tr>
</tbody>
</table>

Poking with uXMS

- uXMS is a command line tool for poking the MOOSDB

```bash
$ uPokeDB mission.moos BAND="abba" ANGLE=45
```

### Before

<table>
<thead>
<tr>
<th>MOOSDB</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>FRUIT</td>
<td>apples</td>
</tr>
<tr>
<td>ANGLE</td>
<td>135</td>
</tr>
<tr>
<td>SPEED</td>
<td>2.8</td>
</tr>
<tr>
<td>NAME</td>
<td>alpha</td>
</tr>
<tr>
<td>WIDTH</td>
<td>86</td>
</tr>
<tr>
<td>HOURS</td>
<td>23</td>
</tr>
</tbody>
</table>

**Poke**

BAND = "abba"

<table>
<thead>
<tr>
<th>MOOSDB</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>FRUIT</td>
<td>apples</td>
</tr>
<tr>
<td>ANGLE</td>
<td>45</td>
</tr>
<tr>
<td>SPEED</td>
<td>2.8</td>
</tr>
<tr>
<td>NAME</td>
<td>alpha</td>
</tr>
<tr>
<td>WIDTH</td>
<td>86</td>
</tr>
<tr>
<td>HOURS</td>
<td>23</td>
</tr>
<tr>
<td>BAND</td>
<td>abba</td>
</tr>
</tbody>
</table>
MOOS Conventions

MOOS Variables are
- Typically uppercase
- seldom use numbers
- Never have white space
- Only special character is the underscore '_'

Nice Variables:
- NAV_HEADING
- TOTAL_POINTS
- DESIRED_SPEED
- CLIENTS

Ugly Variables:
- TIME_OF_DAY
- basic_value
- #ofdays
- SLIP-JOINT

Data Logging
Data Logging

pLogger is a MOOS application that logs all or select publications to a file.

The logger creates at least four files for each mission:
- **file.alog** – asynchronous log (a new entry any time a post is made)
- **file.slog** – synchronous log (a sampling of variable values at fixed intervals)
- **file._bhv** – a log of critical messages
- **file._moos** – a copy of the mission file used to launch the mission.

Log File Format

The alog file format is meant to be human readable.
Configuring the pLogger App

```
ProcessConfig = pLogger
{
    AppTick = 10
    CommsTick = 10
    File = RED_LOG
    PATH = ./
    AsyncLog = true
    FileTimeStamp = true
    // Log it all!!!!!
    LogAuxSrc = true
    WildCardLogging = true
}
```

pLogger, like other MOOS Apps, has a configuration block in mission.moos.

Wildcard Logging with Finer Control
(Exclusion by Pattern Matching)

- Wildcard logging allows you to capture everything
- Variables or variable patterns may be omitted

```
ProcessConfig = pLogger
{
    AppTick = 10
    CommsTick = 10
    File = BLUE_LOG
    PATH = ./
    AsyncLog = true
    FileTimeStamp = true
    WildcardLogging = true
    WildcardOmitPattern = *_STATUS
}
```
Wildcard Logging – Playing it Safe

- What if a variable was excluded by mistake?
- Use the WildcardExclusionLog to log everything otherwise excluded

```
ProcessConfig = pLogger
{
    AppTick = 10
    CommaTick = 10
    File = GREEN_LOG
    PATH = ./
    AsyncLog = true
    SyncLog = true @ 0.2
    FileTimeStamps = true
    WildcardLogging = true
    WildcardOmitPattern = *_STATUS
    WildcardExclusionLog = true
}
```

Will log all MOOS variables ending with: 

```
_STATUS
```
in logfile.xlog

The Alog Toolbox
Tools for Modifying and Analyzing Alog Files

Command-Line log file tools:
- `aloggrep`: Prune an alog file by specifying a set of variables to keep.
- `alogsca,n, aloghelm`: Examine the contents of a alog file in a short summary.
- `alogrm`: Prune an alog file by removing a given set of MOOS variables.
- `alogclip`: Prune an alog file by specifying a min/max timestamp

Each tool is a light-weight single-purpose command-line executable.
Each tool accepts the --help command line option for further usage info.
The aloggrep Tool

- The aloggrep tool is passed an alog file and list of variables to keep
- Output is to the terminal window

```
$ aloggrep file.alog NAV_X NAV_Y
```

- If provided the name of a new alog file, the new file is created
- The new file is a syntactically complete alog file (retaining header info)

```
$ aloggrep file.alog NAV_X NAV_Y newfile.alog
```

Hint: We often use this tool to help us create a focused set of data for debugging.

The alogrm Tool

- The alogrm tool is passed an alog file and list of variables to remove
- Output is to the terminal window

```
$ alogrm file.alog DB_STATUS
```

- If provided the name of a new alog file, the new file is created
- The new file is a syntactically complete alog file (retaining header info)

```
$ alogrm file.alog DB_STATUS newfile.alog
```

Hint: We often use this tool to reduce unnecessary variables to reduce alog file size
The alogclip Tool

- The alogclip tool is passed an alog file and start and end time
- All entries in this time window will be kept.
- Output is to the terminal window

```
$ alogclip file.alog 200 1200
```

- If provided the name of a new alog file, the new file is created
- The new file is a syntactically complete alog file (retaining header info)

```
$ alogclip file.alog 200 1200 newfile.alog
```

Hint: We often use this tool to reduce alog file size

Example alogscan Output

Will report behavior sources on helm output.

Will report multiple sources if applicable.
END