uFldMessageHandler: Handling Incoming Node Messages

March 2022

Michael Benjamin, mikerb@mit.edu
Department of Mechanical Engineering
MIT, Cambridge MA 02139

1 Overview

The uFldMessageHandler application handles incoming messages from a remote MOOSDB. In MOOS, applications "talk" to each other through the normal publish-subscribe means of a common MOOSDB. Distinct robots or vehicles can also share information between vehicles and MOOSDBs using pShare. But often we want to simulate inter-vehicle messaging where messages are sometimes dropped, and messages are subject to inter-vehicle range limitations, band-width limitations, and limitations on message frequency. To simulate this we use the uFldNodeComms and uFldMessageHandler apps in concert. A message meant for another vehicle is packaged up in a format containing the message itself and a bit of routing information. The receiving vehicle, running uFldMessageHandler, unpacks the message and injects the message to the local vehicle by posting to the local MOOSDB. The job of routing and applying communications limitations is done by uFldNodeComms and is not the focus here. The sole job of uFldMessageHandler is to do the message unpacking and posting of information, while also keeping some stats for debugging if needed.

2 Typical Application Topology

In the uField Toolbox typical arrangement, messages arrive from a shoreside MOOS community running uFldNodeComms and pShare as shown below in Figure 1.

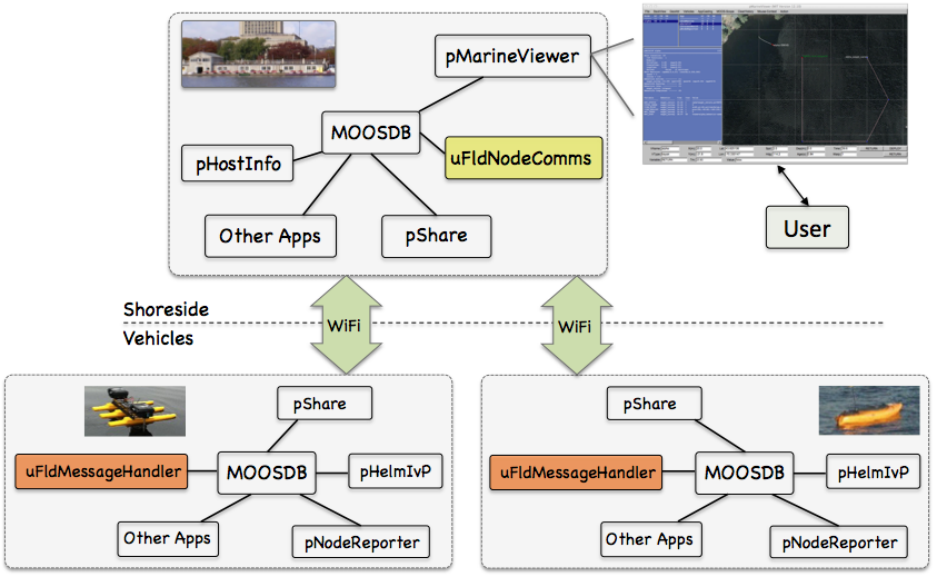

Figure 1: **Typical uFldMessageHandler Topology:** A vehicle (node) sends a message to another vehicle by wrapping the message content and addressee information in a single string sent to the shoreside. On the shoreside, the uFldNodeComms application redirects the message to the appropriate vehicle(s). The message is received on the vehicle by the uFldMessageHandler application which parses the MOOS variable and the variable value from the string and posts the variable-value pair to the local MOOSDB.

3 Inter-vehicle Messaging Sequence of Events

The functionality of `uFldMessageHandler` may be paraphrased:

- A source vehicle *alpha* wishes to send a message to vehicle *bravo* of the form `SPEED=2.5`.
- A local message is posted on vehicle *alpha* of the form:

```
NODE_MESSAGE_LOCAL = src_node=alpha,dest_node=bravo,var_name=SPEED,double_val=2.5
```

- The above message is shared from *alpha* to the shoreside community using `pShare`.
- The message is received in the shoreside community as the variable `NODE_MESSAGE` and handled by `uFldNodeComms` and republished as `NODE_MESSAGE_BRAVO`.
- The message is then shared out to *bravo* using `pShare` arriving in vehicle *bravo* as `NODE_MESSAGE`.
- On vehicle *bravo*, the `NODE_MESSAGE` is handled by `uFldMessageHandler`. The source variable and value are parsed and a post to the local MOOSDB on *bravo* is made, `SPEED=2.5`
- A scope on the MOOSDB on *bravo* would show the source of the `SPEED=2.5` posting to be `"uFldMessageHandler"`, and the auxiliary source would show `"alpha"`

4 Building an Outgoing Node Message

To construct an outgoing node message, there are two options. A message can be created through normal string construction, for example:

```
string m_hostname;      // previously set name of ownship
string m_dest_name;     // previously set name of vehicle to communicate
string m_moos_varname;  // previously set name of MOOS variable to send
string m_msg_contents;  // previously set contents of message;

string msg;
msg += "src_node="   + m_hostname;
msg += ",dest_node=" + m_dest_name;
msg += ",var_name="  + m_moos_varname;
msg += ",string_val=" + m_msg_contents;

Notify("NODE_MESSAGE_LOCAL", msg);
```

The above works, but may be prone to typos and may not be very "future-proof" if the format for inter-vehicle messaging changes. Another way to accomplish this is to use the `NodeMessage` class which has the serializing and de-serializing steps implemented:

```
#include "NodeMessage.h"  // In the lib_ufield library

string m_hostname;     // previously set name of ownship
string m_dest_name;    // previously set name of vehicle to communicate
string m_moos_varname; // previously set name of MOOS variable to send
string m_msg_contents; // previously set contents of message;

NodeMessage node_message;

node_message.setSourceNode(m_hostname);
node_message.setDestNode(m_dest_name);
node_message.setVarName(m_moos_varname);
node_message.setStringVal(m_msg_contents);

string msg = node_message.getSpec();

Notify("NODE_MESSAGE_LOCAL", msg);
```

In the opposite direction, creating a `NodeMessage` instance from a string is done with:

```
#include "NodeMessageUtils.h"  // In the lib_ufield library

string node_message_str;       // previously set string containing a message

NodeMessage node_message = string2NodeMessage(node_message_str);
```

Note that the `NodeMessage` class is part of the `lib_ufield` library. The `uFldMessageHandler` app links to this library, but if the above class is being used in a new app, to create a message, the `lib_ufield` library needs to be linked as part of the build process of the new app.

5 Building an Outgoing Node Message - A Note of Caution

Note the structure of a node message, such as the one below, uses commas as component delimiters:

```
NODE_MESSAGE_LOCAL = src_node=alpha,dest_node=bravo,var_name=SPEED,double_val=2.5
```

Some care must be taken if the contents of the outgoing message is a string also containing commas, such as:

```
NODE_MESSAGE_LOCAL = src_node=alpha,dest_node=bravo,var_name=INFO,string_val=good,bad,ugly
```

In this case the string contents should be wrapped in double quotes:

```
NODE_MESSAGE_LOCAL = src_node=alpha,dest_node=bravo,var_name=INFO,string_val="good,bad,ugly"
```

If the message is built using the `NodeMessage` class as in the second example in Section 4 above, the double quotes are added automatically in the `setStringVal()` function if the argument is not quoted and it contains a comma.

6 Event Flags

With the optional `msg_flag` and `bad_msg_flag`, the user may configure events (postings) to be made whenever an incoming message results in a successful posting, and whenever an incoming message does not result in a posting.

The `msg_flag` is configured with variable-value pair, for example:

```
msg_flag = GOOD_POSTING=true
```

Likewise, the `bad_msg_flag` is configured with variable-value pair, for example:

```
bad_msg_flag = FAILED_POSTING=true
```

The value component of any posting may contain one or more of several supported macros. A macro is expanded at the time of posting. Supported macros include:

- `$[CTR]`: The total of all received messages, regardless of whether a message resulted in a posting.
- `$[GOOD_CTR]`: The total of all received messages that resulted in a posting.
- `$[BAD_CTR]`: The total of all received messages that did not result in a posting.

Of course multiple postings may be configured. For example:

```
msg_flag = GOOD_POSTING=true
msg_flag = POSTING_COUNT=$[CTR]
```

7 Configuration Parameters of uFldMessageHandler

The following parameters are defined for `uFldMessageHandler`.

Listing 7.1: Configuration parameters for `uFldMessageHandler`.

appcast_trunc_msg: Number of characters allowed in the appcast report for each line reporting a successful message. For example, Lines 19-24 in Listing 4. The default is 75. Setting it to zero means no truncating will be applied.

strict_addressing: If true, only messages with a destination specified by `dest_node`, matching the local community name are processed. Other messages with a destination specified by a group designation are ignored. The default is false.

msg_flag: A variable-value pair posted upon receipt of a valid, un-rejected incoming message and posting to the MOOSDB. Section 6.

bad_msg_flag: A variable-value pair posted upon receipt of a valid, but rejected incoming message. Section 6.

aux_info: Adjust the content of the source auxilliary field. By default this is set to `"node"`. If set to `"node+app"`, the source auxialliary field of posted messages will contain both the sending node and the sending app.

An Example MOOS Configuration Block

Listing 2 shows an example MOOS configuration block produced from the following command line invocation:

```
$ uFldMessageHandler --example or -e
```

Listing 7.2: Example configuration of the uFldMessageHandler *application.*

```
1  ============================================================
2  uFldMessageHandler Example MOOS Configuration
3  ============================================================
4
5  ProcessConfig = uFldMessageHandler
6  {
7    AppTick   = 4
8    CommsTick = 4
9
10   strict_addressing = false   // the default
11   appcast_trunc_msg = 75      // default: the number of chars per
12                               // line in the appcasting output
13
14   msg_flag     = RETURN=true
15   bad_msg_flag = TOTAL_BAD=$[BAD_CTR]
16
17   aux_info = node+app  // {node or node+app} Default is node
18
19   app_logging  = true  // {true or file} By default disabled
20 }
```

7.1 Variables Published

The primary output of uFldMessageHandler to the MOOSDB are the messages posted by parsing incoming NODE_MESSAGE postings. A summary is also posted periodically to recap message handling totals.

- APPCAST: Contains an appcast report identical to the terminal output. Appcasts are posted only after an appcast request is received from an appcast viewing utility. Section 9.
- UMH_SUMMARY_MSGS: A summary of total messages, valid messages and rejected messages handled thus far. An example: UMH_SUMMARY_MSGS=total=3,valid=3,rejected=0

Further publications will be made if the app is configure with any msg_flag or bad_msg_flag flags. For example if configured with msg_flag=POSTING=$[CTR], the variable POSTING will be posted each time uFldMessageHhandler posts an incoming message. The value of POSTING will be the total number of messages posted by uFldMessageHandler thus far.

7.2 Variables Subscriptions

The uFldMessageHandler application subscribes to the following MOOS variables:

- APPCAST_REQ: A request to generate and post a new apppcast report, with reporting criteria, and expiration.
- NODE_MESSAGE: Incoming node messages.

8 Command Line Usage of uFldMessageHandler

The uFldMessageHandler application is typically launched with pAntler, along with a group of other vehicle modules. However, it may be launched separately from the command line. The command line options may be shown by typing:

```
$ uFldMessageHandler --help or -h
```

Listing 8.3: Command line usage for the uFldMessageHandler tool.

```
1  ============================================================
2  Usage: uFldMessageHandler file.moos [OPTIONS]
3  ============================================================
4
5  SYNOPSIS:
6  ------------------------------------
7    The uFldMessageHandler tool is used for handling incoming
8    messages from other nodes. The message is a string that
9    contains the source and destination of the message as well as
0    the MOOS variable and value. This app simply posts to the
11   local MOOSDB the variable-value pair contents of the message.
12
13 Options:
14   --alias=<ProcessName>
15       Launch uFldMessageHandler with the given process name
```

```
16        rather than uFldMessageHandler.
17    --example, -e
18        Display example MOOS configuration block.
19    --help, -h
10        Display this help message.
21    --interface, -i
22        Display MOOS publications and subscriptions.
23    --version,-v
24        Display the release version of uFldMessageHandler.
25    --web,-w
26        Open browser to:
27        https://oceanai.mit.edu/ivpman/apps/uFldMessageHandler
28
29  Note: If argv[2] does not otherwise match a known option,
30        then it will be interpreted as a run alias. This is
31        to support pAntler launching conventions.
```

9 Terminal and AppCast Output

The uFldMessageHandler application produces some useful information to the terminal and identical content through appcasting. An example is shown in Listing 4 below. On line 2, the name of the local community or vehicle name is listed on the left. On the right, "0/0(841) indicates there are no configuration or run warnings, and the current iteration of uFldMessageHandler is 841. In lines 4-9, general tallies are shown of received, invalid, and rejected messages. In lines 11-15, the tallies for received messages sorted by source vehicle are shown. The variable-value columns reflect only the last received message.

Listing 9.4: Example appcast and terminal output of uFldMessageHandler*.*

```
 1  ================================================================
 2  uFldMessageHandler gilda                                 0/0(841)
 3  ================================================================
 4  Overall Totals Summary
 5  ======================================
 6     Total Received Valid: 5
 7                    Invalid: 0
 8                   Rejected: 0
 9      Time since last Msg: 101.3
10
11  Per Source Node Summary
12  ======================================
13  Source  Total  Elapsed  Variable  Value
14  ------  -----  -------  --------  -----
15  henry   5      101.3    RETURN    true
16
17  Last Few Messages: (oldest to newest)
18  ======================================
19  Valid Mgs:
20    src_node=henry,dest_node=gilda,var_name=UPDATE_LOITER,string_val=speed
21    src_node=henry,dest_node=gilda,var_name=UPDATE_LOITER,string_val=speed
22    src_node=henry,dest_node=gilda,var_name=UPDATE_LOITER,string_val=speed
23    src_node=henry,dest_node=gilda,var_name=UPDATE_LOITER,string_val=speed
24    src_node=henry,dest_node=gilda,var_name=RETURN,string_val=true
```

```
25  Invalid Mgs:
26    NONE
27  Rejected Mgs:
28    NONE
```

The information group starting on line 17 shows the last five received valid, invalid and rejected messages. Note that a rejected message may be rejected for being invalid, or if the destination field doesn't match, or if strict addressing is enabled and there is not a precise destination field match.