Help Topic: MOOS-IvP Message Handling and Parsing

Spring 2020

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

MOOS-IvP Message Handling and Parsing

There are many books and websites where you find comprehensive information on STL vectors and strings. Here we aim to lay out a quick-start cheat sheet so we can get right to the topic of parsing strings commonly passed as MOOS messages.

MOOS Message Passing Conventions

A common way of sending a MOOS message is to use a comma-separated list of pair=value components. For example:

```
HEALTH_SUMMARY = "temperature=99.1, height=72, weight=175"
```

We could have sent the same message more concisely as follows:

```
HEALTH_SUMMARY = "99.1,72,175"
```

The consumer or recipient of the message could simply have an understanding that this message has three parts with the components indicating *temperature*, *height*, and *weight* respectively. It does make for a shorter message, but it has a few drawbacks: (a) it is not very human readable, (b) the ordering of the components is now important, and (c) if the sender wants to send more information say *age*, or less information because perhaps it doesn't know the weight of the person, then it becomes tricky how to handle this if the receiver is rigidly expecting the original 3-tuple.

A convention, at least in our lab, is to prefer the comma-separated list of parameter=value pairs in the first example above. The message handling routine on the receiver end then can process the compoents in any order, and extra unexpected components can be easily handled or ignored. Message structures tend to *grow* over time. By this we mean for example that a `HEALTH_SUMMARY` message today may only contain the three components above, but somewhere down the road someone may want to add *age*, *gender*, etc., and we don't want any of the original MOOS apps subscribing for this information to break. There are other ways to accomplish this, e.g., Google Protocol Buffers, but the convention we describe here also works pretty well in practice.

So the comma-separated list of parameter=value pairs is pretty easy to construct, and parsing it on the receiver end is also pretty easy but is a bit more involved.

Creating a MOOS Message with STL Strings

By now you have some familiarity with strings in C++. With the advent of the standard template libarry (STL), the most common way of using strings is the STL string class.

```
#include<string>
...
std::string str = "Hello World";
```

The above style MOOS messages are easy to construct:

```
std::string msg = "temperature=99.1, height=72, weight=175";
```

Or, more commonly, if the components are held in local variables, the string message may be built up using string concatenation.

```
string temperature = "99.1";
string height = "72";
string weight = "175";
...
std::string msg;
msg += "temperature=" + temperature + ",";
msg += "height=" + height + ",";
msg += "weight=" + weight;
```

The above works, but what if the *temperature*, *height* and *weight* components were stored locally as a type double instead of string? You will need to first convert the numerical values to string values using `doubleToString`, `intToString`, or `uintToString` functions described on the String Utilities page:

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

For example:

```
double temperature = 99.1;
double height = 72;
double weight = 175;
...
std::string msg;
msg += "temperature=" + doubleToString(temperature,1) + ",";
msg += "height=" + doubleToString(height,0) + ",";
msg += "weight=" + doubleToString(weight,0);
```

Parsing a MOOS Message of Comma-Separated Pairs

Here are a couple ways of parsing and handling an incoming MOOS message with the structure of a comma-separated list of parameter=value pairs. For example:

HEALTH_REPORT = "temperature=98.1, weight=178, height=73"

In the below examples, we assume that an `OnNewMail()` function exists elsewhere that detects an incoming mail message `HEALTH_REPORT`, and invokes the below mail handling routine. We also assume that this fictitous MOOS App has local string member variables, `m_temperature`, `m_weight`, and `m_height`.

```
bool MyMOOSApp::handleMailHealthReport(string report)
{
  vector<string> str_vector = parseString(report, ',');
  for(unsigned int i=0; i<str_vector.size(); i++) {
    string param = biteStringX(str_vector[i], '=');
    string value = str_vector[i];

    if(param == "temperature")
      m_temperature = value;
    else if(param == "height")
      m_height = value;
    else if(param == "weight")
      m_weight = value;

   return(true);
 }
```