Help Topic: MOOS-IvP String Parsing Utilities

Spring 2020

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

MOOS-IvP String Parsing Utilities

The below describe a set of string utilities in the `MBUtils` library distributed with MOOS-IvP. To use them, add `#include "MBUtils.h"` in your source code, and add the `mbutil` library to the list of libraries your code links to, likely in your local `CMakeLists.txt` file.

The `biteString()` function

The `biteString()` function takes a string and returns everything to the left of the given character. The original string is modified and is everything to the right of the character.

```
string biteString(string, char);
```

For example:

```
string orig = "temperature = 98";
string left = biteString(orig, '=');
cout << "left: [" << left << "]" << endl;
cout << "orig: [" << orig << "]" << endl;
```

Produces:

```
left: [temperature ]
orgi: [ 98];
```

Notice that the white space to the left and the right of the '=' character are preserved in the result. If your desire is to have these removed, you can invoke `biteStringX()` instead, described below.

The `biteStringX()` function

The `biteStringX()` function does the same thing as the `biteString` function but takes the additional step of removing blanks from the ends of the results.

```
string biteStringX(string, char);
```

For example:

```
string orig  = "temperature = 98";
string left = biteStringX(orig, '=');
cout << "left: [" << left << "]" << endl;
cout << "orig: [" << orig << "]" << endl;
```

Produces:

```
left: [temperature]
orgi: [98];
```

The `parseString()` function

The `parseString()` function takes a string, and a character. It returns a vector of strings where each string is component of the original string separated by the given character.

```
vector<string> parseString(string, char);
```

For example:

```
string orig = "temperature=98, height=72, weight=150";
vector<string> str_vector = parseString(orig, ',');
for(unsigned int i=0; i<str_vector.size(); i++)
  cout << "component: [" << str_vector[i] << "]" << endl;
```

Produces:

```
 component = [temperature=98]
 component = [ height=72]
 component = [ weight=150]
```

Notice that the white space to the left and the right of the ',' character are preserved in the result.

The `parseStringQ()` function

The `parseStringQ()` works like `parseString()` except that the character separated is ignored if it is enounctered between double-quotes.

```
vector<string> parseStringQ(string, char);
```

For example:

```
string orig = "children="john,bob,mary", height=72, weight=150";
vector<string> str_vector = parseStringQ(orig, ',');
for(unsigned int i=0; i<str_vector.size(); i++)
  cout << "component: [" << str_vector[i] << "]" << endl;
```

Produces:

```
component = [children=john,bob,mary]
component = [ height=72]
component = [ weight=150]
```

Notice that the white space to the left and the right of the ',' character are preserved in the result.

The `tokStringParse()` function

The `tokStringParse()` works on a comma-separated list of parameter=value pairs and pulls out the value for a given parameter. The first character argument is the "global" separator, and the second argument is the "local" separator.

```
string tokStringParse(string, string, char, char);
```

For example:

```
string orig = "temperature=98.1, height=72, weight=150";

string a = tokStringParse(orig, "temperature", ',', '=');
string b = tokStringParse(orig, "height", ',', '=');
string c = tokStringParse(orig, "weight", ',', '=');
string d = tokStringParse(orig, "age", ',', '=');

cout << "a: [" << a << "]" << endl;
cout << "b: [" << b << "]" << endl;
cout << "c: [" << c << "]" << endl;
cout << "d: [" << d << "]" << endl;
```

Produces:

```
a: [98.1]
b: [72]
a: [150]
a: []
```

This function will also strip leading and trailing white space on its return value.