Subject: U++ Command Line Argument Processor Package
Posted by captainc on Mon, 01 Sep 2008 15:30:10 GMT
View Forum Message <> Reply to Message

I made a new package for command line argument parsing. Help me test it out. Let me know how you like it, what's useful and not; any comments, suggestions, etc are appreciated (usefulness, programming style, etc).
Documentation included and simple example packages too.

From srcdoc:
Quote:This program parses arguments and assigns values to them based on argument input. Users define their arguments and conditions, the package will ensure that the arguments are assigned the correct values. Usage information is automatically generated by the argument lists and ready for output to command line (or any Stream).

Philosophy:
Define allowable combinations of arguments and ensure that the arguments provided to the application match in both combination and type.

Features:
- Enforces argument type restrictions.
- Automatically generates usage information for command line input.
- Provides feedback to the user concerning invalid argument input.
- Provides an interface to test for and retrieve argument values and argument sets that were used.

Inspiration:
This U++ package was inspired by TCLAP the Templatized Command Line Parser (tclap.sourceforge.net). Though, TCLAP is not friendly with U++, so it was decided to create similar functionality using U++ types. TCLAP also had some features that restricted it from being used in certain ways. For example, their idea of Xor Arguments caused the usage to be restrictive in combinations. This package differs by defining sets of allowable combinations of arguments instead.

I guess we can call it UCLAP ... !

Example program use (for those lazy to download and test!). I took out the explanation comments from the example file...

```
#include <Core/Core.h>
#include <CmdLineArgProcessor/ArgProcessor.hpp>

using namespace Upp;

CONSOLE_APP_MAIN
{
 const Vector<String>& cmdline = CommandLine();

 ArgProcessor ap("Command Line Argument Processor Test", "0.1");
```

```
 Arg<int> num_lines_arg   ("Number of Lines", "Number of lines to print from each file.");
 Arg<String> file_output_arg  ("o","output-file","Output File", "Write output to file.");
 Arg<bool> print_output_arg  ("p", "print-output", "Print Output to Console", "Print output of files
found to console..");
 Arg<String> file_input_arg   ("File Input", "Single input file to read from.", true);
 MultiArg<String> dir_input_arg ("Directory", "Directory where files are located. (Specify as many
times as necessary).", true);

 ArgSet common_args, file_input_argset, directory_input_argset;
 common_args << num_lines_arg << file_output_arg << print_output_arg;
 file_input_argset << file_input_arg << common_args;
 directory_input_argset << dir_input_arg << common_args;
 ap << file_input_argset << directory_input_argset;

 ap.ProcessCmdLine(cmdline);

 if(!ap.IsError())
 {
  if(file_input_argset.IsMatch())
  {
   Cout() << "File input argset matched, File input arg: " << file_input_arg.Val() << "\n";
  }
  else if(directory_input_argset.IsMatch())
  {
   Cout() << "Directory list input argset matched.\n";
   for(int i=0; i<dir_input_arg.GetCount(); i++)
   {
    Cout() << "\tDirectory: " << dir_input_arg[i].ToString() << "\n";
   }
  }
  if(print_output_arg.IsSet())
  {
   Cout() << "Print output argument specified.\n";
  }
  if(file_output_arg.IsSet())
  {
   Cout() << "File output argument specifed: " << file_output_arg.Val() << "\n";
  }
  if(num_lines_arg.IsSet())
  {
   int num_lines = num_lines_arg.Val(); // implicit conversion from Value to int.
   Cout() << "Number of lines argument specified: " << num_lines << " lines.\n";
  }
 }
 else
 {
  Cout() << ap.GetError() << "\n";
  ap.PrintUsageInformation();
```

```
 }
}
```

The Test and Example packages are similar; the test just tests functionality and the example puts it to use in a simple application.

Update 9/06/2008: Replaced with v0.2 ... Change log in T++.
Update 9/29/2008: Replaced with v0.2.1
-> Moved all template code for Arg and TArg to header file to eliminate potential pesky unresolved external errors.
Update 11/15/2008: Version 0.3.0 Released (See Below)

File Attachments

1) CmdLineArgProcessor-0.2.1-All.zip, downloaded 787 times

Subject: Re: U++ Command Line Argument Processor Package
Posted by mr_ped on Mon, 01 Sep 2008 15:35:57 GMT
View Forum Message <> Reply to Message

sounds interesting, I will try to test it soon, I hope I will not forget.

Subject: Re: U++ Command Line Argument Processor Package
Posted by captainc on Sat, 15 Nov 2008 17:26:03 GMT
View Forum Message <> Reply to Message

All new version of UCLAP. Version 0.3.0.
This is a major upgrade to support validation for standard arguments as well as delimited arguments.
Backwards compatibility with 0.2.x series is broken.

Programs called take the form:MyProgramExecutable [OPTIONS] [ARGUMENTS]
Where:
OPTIONS is an unordered list of command line options specified by delimiters and flags
ARGUMENTS is an ordered list of arguments that come after the OPTIONS

Example: the tail program output the last few lines of a file.
tail -n 3 myfile

-n is an option (delmited argument) whose value is 3.
myfile is a normal base argument.

So now, in UCLAP, you specify options and arguments like this:
(From example 3b in package)
ArgProcessor ap("CmdLineArgProcessor Example", "3b");
Option<bool> sum_opt ("Summation", "If specified, program will calculate the sum of the

```
numbers.", true);
Option<bool> avg_opt ("Average", "If specified, program will calculate the average of the
numbers.", true);
MultiArg<double> numbers ("Number", "An number.", true);
ArgSet set1_sum, set2_avg;
set1_sum << sum_opt << numbers;
set2_avg << avg_opt << numbers;
ap << set1_sum << set2_avg;
ap.ProcessCmdLine(CommandLine());
```

And the usage information generated for this code above looks like this:

```
Usage Information:
1. CmdLineArgProcessorExample -s <Number1> ...
2. CmdLineArgProcessorExample -a <Number1> ...
    -s, --summation :  If specified, program will calculate the sum of the numbers.
    -a, --average   :  If specified, program will calculate the average of the numbers.
    Number :  An number.
        This argument must be specified 1 to an unlimited number of times.
```

Todo:
- Work a bit more on output formatting (maybe make an output formatting package).
--> most of the gnu tools limit the line width to 80 characters for all programs that output to the command line. Small consoles might be a problem for the output of this package in certain situations.
- See if using an Xor Argument type would be beneficial and shorten the output.

The zip file contains the main package, a package with examples, and the unit testing package. Topic++ documentation has also been updated.

File Attachments

1) CmdLineArgProcessor-0.3.0-all.zip, downloaded 727 times

Subject: Re: U++ Command Line Argument Processor Package
Posted by mdelfede on Wed, 25 Jan 2017 18:02:56 GMT
View Forum Message <> Reply to Message

Hi, I guess that a bugfix is needed, as it is if ONE option matches the whole stuff do, even if some are missing.
I replaced ArgProcessor::MatchesOptions with the following :

```
bool ArgProcessor::MatchesOptions(Ptr<ArgSet> arg_set)
{
 // check for required options
```

```
 for(int i=0; i<arg_set->_options.GetCount(); i++)
 {
  if(arg_set->_options[i]->IsRequired())
  {
   bool found = false;

   for(int j=0; j<_arg_list.GetCount(); j++)
   {
    if(arg_set->_options[i] == _arg_list[j])
    {
     if(arg_set->_options[i]->IsSet())
     {
      found = true;
      break;
     }
    }
   }
   if(!found)
    return false;
  }
 }

 // check for spurious options
 for(int j=0; j<_arg_list.GetCount(); j++)
 {
  bool found = false;
  for(int i=0; i<arg_set->_options.GetCount(); i++)
  {
   if(arg_set->_options[i] == _arg_list[j])
   {
    found = true;
    break;
   }
  }
  if(!found)
   return false;
 }
 return true;
}
```

No time to check if it solves all problems, but in my case it's ok.
Now it's checking that ALL required options are there and that NO un-listed option is there.

Ciao

Massimo
