NAME

Class CommandOptions − handle command-line options using GNU **getopt_long**(3) along with help text for each option.

SYNOPSIS

#include <CommandOptions.h>

CommandOptions();
CommandOptions(const CommandOptions& *opt***);**
~CommandOptions();
void setOption(const char* *shortName***, const char*** *longName***, const char*** *desc***);**
void setOption(const char* *shortName***, const char*** *longName***, const char*** *desc***,**
const int *type***, const char*** *argument***);**
void setOption(const int *value***, const char*** *longName***, const char*** *desc***);**
void setOption(const int *value***, const char*** *longName***, const char*** *desc***,**
const int *type***, const char*** *argument***);**
void done();
int getOption(int *argc***, char*** *argv[]***);**
int getNumberOfOptions();
int getOptionCount();
int getOptionError();
int getOptionOpt();
char* getOptionArgument();
void quiet();
void verbose();
void showOptionsHelp();
void showOptionsHelp(int *width***);**

DESCRIPTION

It is desirable for programs accepting command-line options to provide a description of the options when requested. Normally **getopt**(3) or the GNU function **getopt_long**(3) are used to process command-line options with a *while*-loop and a *switch* block. There is no mechanism, however, to synchronize this code with usage information displayed by the program. The **CommandOptions** class provides a simplified interface to **getopt_long**(3) and allows a programmer to specify the help text for each option at the time the other option information is defined.

The class constructor **CommandOptions**() must be called first. Options are then installed one-by-one using the **setOption**() methods. If the first argument to **setOption**() is a *char* then it specifies a valid short option; if it is an *int* then only the long option name will be used. The second and third options are the long option name and the help text for the option. If fourth and fifth arguments are specified then they specify the type and name of the option's argument; *type* can be **CommandOptions::None** (equivalent to not supplying last two arguments), **CommandOptions::Required**, or **CommandOptions::Optional**. The method **done**() must be called to finalize the option list before calls to **getOption**() are made.

The work of processing the option list is done by **getOption**(). This method is used similarly to how **getopt**(3) and **getopt_long**(3) are used. Its arguments, *argc* and *argv* are the argument count and array passed to the main function when the program is started. Each call to **getOption**() causes another option to be processed and either the option's short name character or, in the case of long-only options, the options integer value, is returned. When no more options are available the method returns -1.

The **getopt.h** file is implicitly included by **CommandOptions.h** so the four external global variables *optarg*, *opterr*, *optopt*, and *optind* are available to programs using this class. Users are encouraged, however, to use the corresponding accessor methods provide by this class. The values of the first three variables can be accessed with **getOptionArgument**(), **getOptionError**(), and **getOptionOpt**(). After all options have been processed the variable *optind* holds the index of the first non-option parameter in the *argv* array (the array has already been sorted so that options and their arguments appear before all non-option

parameters). Rather than accessing this variable directly, the method **getOptionCount**() returns *optind*-1, which gives the number of options. This value can be used to adjust both *argc* and *argv* if desired.

The accessor **getNumberOfOptions**() returns the number of options that have been defined thus far. The mutators **quiet**() and **verbose**() change the value of *opterr*. If *opterr*=0 then error messages from **getopt_long**(3) are suppressed.

The method **showOptionsHelp**() uses **iostreams** to write the option names and descriptions to the output stream. The optional parameter *width* sets the width in characters of the option name portion of the output; the default width is 35 characters.

EXAMPLE

The following code will create a CommandOptions object:

```
enum {ShowHelp, ShowVersion};

CommandOptions options;
options.setOption( 's', "server", "server to display data for",
                     CommandOptions::Optional, "SERVER" );
options.setOption( 'w', "write-file", "write data to file",
                     CommandOptions::Required, "FILE" );
options.setOption( 'q', "quiet", "show minimum information" );
options.setOption( 'v', "verbose", "show maximum information" );
options.setOption( ShowVersion, "version", "display version" );
options.setOption( ShowHelp, "help", "show this help message" );
options.done();
```

Assuming the functions **version**() and **help**() have been defined, the object can now be processed with code like

```
char* programName = argv[0];

bool quiet = false;
bool verbose = false;
bool writeFile = false;
bool useServer = false;
string fileName;
int server = 1; // default server number

int ch;
while ((ch = options.getOption(argc, argv)) >= 0)
{
    switch (ch)
    {
        case 'w':
            writeFile = true;
            fileName = options.getOptionArgument();
            break;
        case 's':
            useServer = true;
            if (options.getOptionArgument() != NULL)
                server = atoi(options.getOptionArgument());
            break;
        case 'q':
            quiet = true;
            break;
```

```
                case 'v':
                    verbose = true;
                    break;
                case ShowHelp:
                    help( programName, options );
                    return EXIT_SUCCESS;
                case ShowVersion:
                    version( programName );
                    return EXIT_SUCCESS;
                    break;
                case '?':
                default:
                    help( programName, options );
                    return EXIT_FAILURE;
            }
        }

        // adjust the argument count and array to skip over
        // the options we've just processed
        argc -= options.getOptionCount();
        argv += options.getOptionCount();
```

The **help**() function might look like

```
    void help(const char* name, CommandOptions& options)
    {
        cout << "usage: " << name << " [OPTIONS] INPUT" << endl;
        cout << endl << "OPTIONS may be one or more of" << endl;
        options.showOptionsHelp();
    }
```

SEE ALSO

getopt(3), **getopt_long**(3)

AUTHOR

Jonathan Senning