clo++ Users Manual

Peter Jones on October 30, 2007

Contents

List of Tables

Abstract

This is the manual for clo++. It describes how to use clo++ and the format of the XML input file.

The latest version of this manual can be found at http://pmade.org/software/clo++/.

Chapter 1

Introduction

1.1 What is clo++?

clo++ is a command line option parser generator. Given an XML file that contains a description of your program and its options, clo++ can generate code to parse its command line.

In addition to code generation, clo++ can also generate man pages. Other languages and output formats can easily be added because clo++ generates its output using templates.

1.2 Features

clo++ tries to provide enough features to parse most forms of command line options without going overboard. In most cases, if clo++ does not support a specific feature you are looking for, you can use an existing feature and customize it in your application.

The following list is just a quick overview of the feature list for clo++.

- Option description is done using a simple XML dialect instead of a complex proprietary language.
- The users of your application do not have to have clo++ installed. Just include the output from clo++ with your software packages. The clo++ output does not have any license or restrictions.
- Automatic generation of help messages.
- Single character options start with a dash ('-') and multi-character options start with a double dash ('--');
- Support for multiple sub-commands that take their own options. (Similar to cvs commit, cvs add, etc.)
- Multi-character options and commands can be abbreviated depending on a configuration setting in the XML file.
- Single character options can be bundled. (-aeiou).
- Multi-character options can include their argument in the option name or it can be separate. (--option=one --option two).
- Options can have default values. You can see if the value of an option is the result of its default value or its presence on the command line by checking its location value.
- Options can be hidden. That is, they can be used on the command line but will not be shown in the help message or man page.
- Options and Groups can be mandatory. This means that the option/group is required to be used.
- Options can be strict. This means that the option can not be used on the command line more than once. If an option is not strict, it will be allowed on the command line more than once and the last time it is used it the final value of the option.
- The clo++ source code is released under a BSD license. There are very few restrictions on its use.

1.3 Supported Option Types

clo++ supports the following option types.

- flag - This option takes no arguments and simply sets a boolean value to indicate its presence on the command line.
- bool - This option takes an argument that is parsed as a boolean value. You can define words that you consider true and false in your XML file.
- enum - This option takes an argument that must be in a list of allowed words. The parser then converts the argument into an enum member for you.
- int - This option takes an argument that must be completely convertible to an integer. You may also specify a minimum and/or a maximum value in your XML file.
- double - This option takes an argument that must be completely convertible to a real number. You may also specify a minimum and/or a maximum value in your XML file.
- string - This option takes an argument that is not parsed and is just stored as a raw character string.

1.4 Option Modifiers

All options except the flag option can have modifiers. Here is a quick overview of the modifier types.

- vector - Each time the option is repeated on the command line its value is pushed into a vector.
- map - The option can take key=value pairs where the key is a string and the value is the value type for the option. This also implies that the option can be repeated.

1.5 Option Grouping

clo++ also supports option grouping. There are three types of groups that an option can be part of. Here is a quick description of them.

- and - This group allows you to enforce the presence of all contained options if one of the options in the group is used.
- or - This group is used to say that at least one option in the the group must be given, but more than one is acceptable.
- xor - This group is used when you want to ensure that a group of options can not be used at the same time.

Chapter 2

Quick Start

We start off with a quick start guide. This is just a fast overview to get you working with clo++ as soon as possible.

2.1 Building and Installing

You build clo++ using the `configure.pl` perl script that comes in the tarball. If you have the `xml2-config` script that comes with libxml2 in your path you can just run `configure.pl`.

If `xml2-config` is not in your path, you can use the **--xml2-config** option to `configure.pl`.

After you run `configure.pl` you are ready to run **make**. Then, after everything compiles, you can run **make install**.

2.2 Writing Your First clo++ XML File

Before you can run clo++, you should have an XML file ready. The XML file should describe the command line options that you wish to parse.

A good place to start is the `examples` directory. There are some example XML files and C++ code that uses the generated C++ code. You may also want to look at the DTD, it is in the `dtd` directory.

2.3 Running clo++

After you have an XML file all ready to go, you can run clo++. I assume that you want to generate C++ code, so we are going to use the **-o c++** option to clo++.

If your XML file is called `options.xml` then this is the command that you would run to generate the C++ header file and source file.

```
clo++ -o c++ options.xml
```

To learn about the other command line options that clo++ can take, try running **clo++ -h**.

Chapter 3

Option Types

This chapter deals with the various option types. Options are expressed in the XML file using `<option>` elements.

Each option has optional and mandatory attributes. It also has some optional and mandatory elements. Most options share the same attributes and elements.

Attribute Name	Optional or Mandatory	Description	Valid Values	Default Value
type	Mandatory	The option type	flag, bool, enum, int, double or string	None, you must give an option type
id	Mandatory	Unique ID	An alpha-numeric value	None, you must give an option id
mandatory	Optional	Whether or not this option must be used	yes or no	no
strict	Optional	Strict options cannot be repeated	yes or no	no
location	Optional	Allowed locations where the option can be used	commandline, configfile or both	both
hidden	Optional	Whether or not this option should be hidden from help messages and man pages	yes or no	no

Table 3.1: Common Option Attributes

3.1 Flag Option Type

The flag option does not take an argument, thus it is used only to know if the option is on the command line. The first time it appears on the command line it will negate the default value. Each time it appears after that it will negate the last value.

To disable this negation behavior, make the option strict. this will prevent the option from repeating.

Unlike other option types, flag options cannot use modifiers. Flag options can, like all options, be mandatory, although it does not make much sense.

Flag options can also be autothrow options.

Example:

Element Name	Description
name	Option name. This element may be repeated for each name you want to give to an option. Single character names make short options (ie, -a) and multi-character names make long options (ie, --alpha). The first The first name given will be the master name. The master name is the one presented to the user in error and help messages.
comment	Small description of the option that is used in help messages.
description	Long description of the option. The description is not given directly in the `<description>` element but rather in child elements called `<para>` similar to DocBook.

Table 3.2: Common Option Child Elements

Attribute Name	Optional or Mandatory	Description	Valid Values	Default Value
default	Optional	The initial value of the option.	A valid boolean value	false
autothrow	Optional	Make this option an autothrow option.	Unique alpha-numeric ID	None, you must give an id

Table 3.3: Flag Option Attributes

```
<option id="flagid" type="flag">
    <name>myflag</name>
    <name>m</name>

    <comment>my flag option</comment>
</option>
```

3.2 Bool Option Type

The bool type option is used to set a boolean value from a string argument. The given argument will be tested against allowed words for true and false. Words that are not recognized as being true or false will not be allowed.

You can control which words are considered true or false by using the `<true>` and `<false>` elements inside the `<config>` element.

Attribute Name	Optional or Mandatory	Description	Valid Values	Default Value
default	Optional	The initial value of the option	A valid boolean value	false
modifier	Optional	The modifier to use, if any	none, vector or map	none
argname	Optional	The string to display for the name of the argument	A short string	bool

Table 3.4: Bool Option Attributes

Example:

```
<option id="boolid" type="bool">
    <name>mybool</name>
    <name>m</name>

    <comment>my bool option</comment>
</option>
```

3.3 Enum Option Type

The enum type option takes a string argument that must match one of the allowed strings for the given option. The string is then translated into an enum member.

Example:

```
<option id="fruit" type="enum" argname="fruit">
    <name>fruit</name>
    <name>f</name>

    <enum id="apl" name="apple"/>
    <enum id="org" name="orange"/>
    <enum id="str" name="strawberry"/>
```

Attribute Name	Optional or Mandatory	Description	Valid Values	Default Value
default	Optional	The initial value of the option	The name of one of the enum members	The first defined enum member
modifier	Optional	The modifier to use, if any	none, vector or map	none
argname	Optional	The string to display for the name of the argument	A short string	enum

Table 3.5: Enum Option Attributes

Element Name	Description
enum	The enum element is used to add a member to the enum. It has two attributes. The `id` attribute will be appended to the option id to create the enum member name. The `name` attribute is the name the user will use on the command line for this member.

Table 3.6: Enum Option Child Elements

```
    <comment>give a fruit you want me to eat</comment>
</option>
```

3.4 Int Option Type

The int option takes a string argument that is converted to an integer. The the entire string cannot be converted to an integer, an error will occur.

Attribute Name	Optional or Mandatory	Description	Valid Values	Default Value
default	Optional	The initial value of the option	A valid integer value	0
modifier	Optional	The modifier to use, if any	none, vector or map	none
argname	Optional	The string to display for the name of the argument	A short string	int

Table 3.7: Int Option Attributes

Example:

```
<option id="age" type="int">
    <name>age</name>
    <name>a</name>

    <range min="0" max="150"/>
```

Element Name	Description
range	The range element allows you to restrict the value of the given integer using either a minimum value, maximum value or both. The `min` attribute sets the minimum value that will be accepted. The `max` attribute sets the maximum value that will be accepted.

Table 3.8: Int Option Child Elements

```
    <comment>give your age</comment>
</option>

<option id="count" type="int">
    <name>count</name>
    <name>c</name>

    <range min="0"/>

    <comment>give the number of times to repeat</comment>
</option>

<option id="number" type="int">
    <name>number</name>
    <name>n</name>

    <comment>give a number</comment>
</option>
```

3.5 Double Option Type

The double option is similar to the int option. It takes a string argument that is converted into a real number. If the entire string cannot be converted and error will occur.

Attribute Name	Optional or Mandatory	Description	Valid Values	Default Value
default	Optional	The initial value of the option	A valid real number	0.0
modifier	Optional	The modifier to use, if any	none, vector or map	none
argname	Optional	The string to display for the name of the argument	A short string	double

Table 3.9: Double Option Attributes

Example:

```
<option id="mydouble" type="double">
    <name>mydouble</name>
```

Element Name	Description
range	The range element allows you to restrict the value of the given double using either a minimum value, maximum value or both. The `min` attribute sets the minimum value that will be accepted. The `max` attribute sets the maximum value that will be accepted.

Table 3.10: Double Option Child Elements

```
    <name>m</name>

    <comment>give me a double</comment>
</option>
```

3.6 String Option Type

The string option takes a string argument that is not parsed in any way.

Attribute Name	Optional or Mandatory	Description	Valid Values	Default Value
default	Optional	The initial value of the option	A string value	An empty string
modifier	Optional	The modifier to use, if any	none, vector or map	none
argname	Optional	The string to display for the name of the argument	A short string	string

Table 3.11: String Option Attributes

Example:

```
<option id="first" type="string">
    <name>first</name>
    <name>f</name>

    <comment>give your first name</comment>
</option>
```

Appendix A

Requirements

clo++ should compile on any platform that has a working ISO 14882-1998 compliant compiler. Okay, so there is no such thing, but your compiler should be pretty close.

No platform specific code is used anywhere in clo++, including the code that it generates. All code is pure C++.

Warning
clo++ will not compile if you are using any version of GCC prior to version 3.0 unless you use the STLport. Furthermore, it will not operate correctly if compiled under GCC versions between 3.0 and 3.0.3 (due to bugs with exceptions).

A.1 Software Dependencies

clo++ has dependencies on the following software.

- Perl 5 - Only needed to configure clo++. Perl is not used any other time.
- libxml2 - The XML parser that clo++ uses. Other parsers will be supported in the future.

A.2 Supported Platforms

clo++ is supported on the following platforms with the given compilers. If you successfully compile clo++ on a platform that is not listed here, please drop us a note.

Platform	Operating System	Compiler
IA32	FreeBSD versions 4.4 - 4.6	GCC versions 3.0.3 - 3.2
IA32	Linux (RH 7.3)	GCC version 3.2
SPARC	Solaris 2.7	GCC version 3.0.4

Table A.1: Platform List

Appendix B

Revision History

B.1 Version 0.6.4

- Made a change to the generated code to surpress a compiler warning about an unused variable.
- Added the cxx_include variable so you can have the generated C++ header file include other header files.
- Added xsl/clo++2html.xsl to show how to convert the clo++ XML file to HTML.
- Added work around in build scripts for a bug in the Perl 5.8 regex parser thanks to Andy Chou.
- Updated xmlwrapp to version 0.4.2 and libtpt to version 1.20.1a.

B.2 version 0.6.3

Version 0.6.3 is a bug fix release.

- Upgraded xmlwrapp to version 0.4.1. clo++ should compile against libxml2 version >= 2.5 now.
- Upgraded libtpt to version 1.20 with a custom patch to fix a bug on 64bit platforms.
- Fixed a few bugs that caused bad C++ code to be generated for complex subcommands.

B.3 version 0.6.2

the following changes where made to clo++ for version 0.6.2.

- Added support for the Sun Forte compiler using STLport.
- upgraded libtpt to version 1.10.5.

B.4 version 0.6.1

the following changes where made to clo++ for version 0.6.1.

- added missing include of the <iterator> header file in two source files (thanks goes to doug henry).

- changed the example makefiles so that gnumake would know that the .cxx file extension is for c++ files. (thanks goes to richard booth).
- added rpm spec file in build/rpm. (thanks goes to warwick hunter).
- upgraded libtpt to version 1.0.1 and xmlwrapp to version 0.2.2.

B.5 Version 0.6.0

The following changes where made to clo++ for version 0.6.0.

- clo++ can now generate man pages.
- Modified clo++.dtd. Added elements to the program element for man page generation. Removed the program/description element and replaced it with the new program/heading element.
- Added new XSLT file to convert old clo++ version 0.3.0 XML files to the new XML format.
- Added another chapter to the documentation.
- Upgraded libtpt to 0.9.5.

B.6 Version 0.5.0

Version 0.5.0 was a complete rewrite.

B.7 Version 0.4.0

Version 0.4.0 was never released to the public.

B.8 Version 0.3.0

First public version.

Appendix C

To Do List

C.1 Feature To Do List

- Allow automap to automatically place non-options that look like a key=value pair into the given map.
- Allow autovector to place non-options into the given vector instead of the default non-options vector.
- Allow commands to be hidden similar to hiding options.
- Add a new option type, count. This type is similar to flag but gets incremented each time it is present on the command line.
- Add an option to allow a '+' to start a long option.
- Find some way to skip unknown options in a safe way. Stuff like '-display' for GTK+ programs.
- Allow enum arguments to be abbreviated.
- Add config flag to make things case-insensitive.
- Don't allow C++ keywords to be used as option IDs when generating C++ code. Also, make sure that autohelp names are not used as names for other options.
- Add member functions to the main parser class for getting the help strings.

C.2 Milestone 0.7.0

- Finish users manual (XML format and C++ API)
- Fix any bugs.
- Implement any requested features. Implement the items from the "Features To Do List".

C.3 Milestone 0.8.0

- Finish template manual so others can write templates.
- Fix any bugs.
- Implement any requested features.

C.4 Milestone 0.9.0

- Code freeze. Make stable.

C.5 Milestone 1.0.0

- First stable release.

Appendix D

Credits

This section is dedicated to those people who contributed in some way to clo++. If I have forgotten you, please forgive me and drop me a note.

D.1 Continued Contributions

The following people make on-going contributions to clo++

- Peter Jones
  - Project maintainer and lead developer.
- Isaac Foraker
  - Author of the template system that clo++ uses.
  - Likes to report bugs.
- Warwick Hunter
  - Contributed an RPM spec file.

D.2 Contributions to Version 0.3.0

The following people either contributed patches or suggestions to clo++ version 0.3.0. Since version 0.3.0 was not supported, their hard work was never applied to the source code.

Although their contributions where not used in clo++, these are the people that inspired clo++ version 0.5.0.

- Matthias Berse
- Christophe de Vienne
- Diab Jerius