NAME

argtable2 − an ANSI C library for parsing GNU style command line options

SYNOPSIS

```
#include <argtable2.h>
```

struct **arg_lit**
struct **arg_int**
struct **arg_dbl**
struct **arg_str**
struct **arg_rex**
struct **arg_file**
struct **arg_date**
struct **arg_rem**
struct **arg_end**

struct **arg_lit*** **arg_lit0**(const char *shortopts, const char *longopts, const char *glossary)
struct **arg_lit*** **arg_lit1**(const char *shortopts, const char *longopts, const char *glossary)
struct **arg_lit*** **arg_litn**(const char *shortopts, const char *longopts, int mincount, int maxcount, const char *glossary)

struct **arg_int*** **arg_int0**(const char* shortopts, const char* longopts, const char* datatype, const char* glossary)
struct **arg_int*** **arg_int1**(const char *shortopts, const char *longopts, const char* datatype, const char *glossary)
struct **arg_int*** **arg_intn**(const char *shortopts, const char *longopts, const char* datatype, int mincount, int maxcount, const char *glossary)

struct **arg_dbl*** **arg_dbl0**(const char *shortopts, const char *longopts, const char* datatype, const char *glossary)
struct **arg_dbl*** **arg_dbl1**(const char *shortopts, const char *longopts, const char* datatype, const char *glossary)
struct **arg_dbl*** **arg_dbln**(const char *shortopts, const char *longopts, const char* datatype, int mincount, int maxcount, const char *glossary)

struct **arg_str*** **arg_str0**(const char *shortopts, const char *longopts, const char* datatype, const char *glossary)
struct **arg_str*** **arg_str1**(const char *shortopts, const char *longopts, const char* datatype, const char *glossary)
struct **arg_str*** **arg_strn**(const char *shortopts, const char *longopts, const char* datatype, int mincount, int maxcount, const char *glossary)

struct **arg_rex*** **arg_rex0**(const char* shortopts, const char* longopts, const char* pattern, const char* datatype, int flags, const char* glossary)
struct **arg_rex*** **arg_rex1**(const char* shortopts, const char* longopts, const char* pattern, const char* datatype, int flags, const char* glossary)
struct **arg_rex*** **arg_rexn**(const char* shortopts, const char* longopts, const char* pattern, const char* datatype, int mincount, int maxcount, int flags, const char* glossary)

struct **arg_file*** **arg_file0**(const char* shortopts, const char* longopts, const char* datatype, const char* glossary)
struct **arg_file*** **arg_file1**(const char *shortopts, const char *longopts, const char* datatype, const char *glossary)
struct **arg_file*** **arg_filen**(const char *shortopts, const char *longopts, const char* datatype, int mincount, int maxcount, const char *glossary)

struct **arg_date*** **arg_date0**const char* shortopts, const char* longopts, const char* format, const char* datatype, const char *glossary)
struct **arg_date*** **arg_date1**const char* shortopts, const char* longopts, const char* format, const char* datatype, const char *glossary)
struct **arg_date*** **arg_daten**const char* shortopts, const char* longopts, const char* format, const char* datatype, int mincount, int maxcount, const char *glossary)

struct **arg_rem*** **arg_rem**(const char *datatype, const char *glossary)
struct **arg_end*** **arg_end**(int maxerrors)

int **arg_nullcheck**(void **argtable)
int **arg_parse**(int argc, char **argv, void **argtable)
void **arg_print_option**(FILE *fp, const char *shortopts, const char *longopts, const char *datatype, const char *suffix)
void **arg_print_syntax**(FILE *fp, void **argtable, const char *suffix)
void **arg_print_syntaxv**(FILE *fp, void **argtable, const char *suffix)
void **arg_print_glossary**(FILE *fp, void **argtable, const char *format)
void **arg_print_glossary_gnu**(FILE *fp, void **argtable)
void **arg_print_errors**(FILE *fp, struct arg_end *end, const char *progname)
void **arg_freetable**(void **argtable, size_t n)

DESCRIPTION

Argtable is an ANSI C library for parsing GNU style command line arguments with a minimum of fuss. It enables the programmer to define their program's argument syntax directly in the source code as an array of structs. The command line is then parsed according to that specification and the resulting values stored directly into user−defined program variables where they are accessible to the main program.

This man page is only for reference. Introductory documentation and example source code is typically installed under **/usr/local/share/doc/argtable2/** and is also available from the argtable homepage at http://argtable.sourceforge.net.

Constructing an arg_<xxx> data structure

Each **arg_<xxx>** struct has it own unique set of constructor functions (defined above) which are typically of the form:

struct **arg_int* arg_int0**("f", "foo", "<int>", "the foo factor")
struct **arg_int* arg_int1**("f", "foo", "<int>", "the foo factor")
struct **arg_int* arg_intn**("f", "foo", "<int>", 2, 4, "the foo factor")

where arg_int0() and arg_int1() are merely abbreviated forms of arg_intn(). They are provided for convenience when defining command line options that have either zero−or−one occurrences (mincount=0,maxcount=1) or exactly one occurrence (mincount=1,maxcount=1) respectively.

The shortopts="f" parameter defines the option's short form tag (eg -f). Multiple alternative tags may be defined by concatenating them (eg shortopts="abc" defines options −a, −b and −c as equivalent). Specify shortopts=NULL when no short option is required.

The longopts="foo" parameter defines the option's long form tag (eg −−foo). Multiple alternative long form tags may be separated by commas (eg longopts="size,limit" defines −−size and −−limit). Do not include any whitespace in the longopts string. Specify longopts=NULL when no long option is required.

If both shortopts and longopts are NULL then the option is an untagged argument.

The datatype="<int>" parameter is a descriptive string that denotes the argument data type in error messages, as in −−foo=<int>. Specifying datatype=NULL indicates the default datatype should be used. Specifying datatype="" effectively disables the datatype display.

The mincount=2 and maxcount=3 parameters specify the minimum and maximum number of occurrences of the option on the command line. If the command line option does not appear the required number of times then the parser reports a syntax error.

The glossary="the foo factor" parameter is another descriptive string. It appears only in the glossary table that is generated automatically by the **arg_print_glossary** function (described later).

```
-f, --foo=<int>    the foo factor
```

Specifying a NULL glossary string causes that option to be omitted from the glossary table.

LITERAL COMMAND LINE OPTIONS

−x, −y, −z, −−help, −−verbose

struct **arg_lit**
  {
  struct **arg_hdr** hdr; /* internal argtable header */
  int count;   /* number of matching command line options */
  };

Literal options take no argument values. Upon a successful parse, *count* is guaranteed to be within the *mincount* and *maxcount* limits specified at construction.

INTEGER COMMAND LINE OPTIONS

−x2, −z 32MB, −−size=734kb, −−hex 0x7, −−binary 0b10011010, −−octal 0o123

Argtable accepts command line integers in decimal (eg 123), hexadecimal (eg 0xFF12), octal (eg 0o123) and binary (eg 0b0101110) formats. It also accepts integers that are suffixed by "KB" (x1024), "MB" (x1048576) or "GB" (x1073741824). All characters are case insensitive

struct **arg_int**
  {

```
  struct arg_hdr hdr;  /* internal argtable header */
  int count;           /* number of values returned in ival[] */
  int *ival;           /* array of parsed integer values */
  };
```

Upon a successful parse, *count* is guaranteed to be within the *mincount* and *maxcount* limits set for the option at construction with the appropriate values store in the *ival* array. The parser will not accept any values beyond that limit.

Hint: It is legal to set default values in the *ival* array prior to calling the **arg_parse** function. Argtable will not alter *ival* entries for which no command line argument is received.

Hint: *Untagged* numeric arguments are not recommended because GNU getopt mistakes negative values (eg -123) for tagged options (eg -1 -2 -3). *Tagged* arguments (eg -x -123, --tag=-123) do not suffer this problem.

REAL/DOUBLE COMMAND LINE OPTIONS

−x2.234, −y 7e−03, −z−3.3E+6, −−pi=3.1415, −−tolerance 1.0E-6

```
struct arg_dbl
  {
  struct arg_hdr hdr;  /* internal argtable header */
  int count;           /* number of values returned in dval[] */
  double *dval;        /* array of parsed double values */
  };
```

Same as **arg_int** except the parsed values are stored in *dval* as doubles.

STRING COMMAND LINE OPTIONS

−Dmacro, −t mytitle, −m "my message string", −−title="hello world"

```
struct arg_str
  {
  struct arg_hdr hdr;  /* internal argtable header */
  int count;           /* number of strings returned in sval[] */
  const char **sval;   /* array of pointers to parsed argument strings */
  };
```

Same as **arg_int** except pointers to the parsed strings are returned in *sval* rather than a separate copy of the string. Indeed, these pointers actually reference the original string buffers stored in argv[], so their contents should not be altered. However, it is legal to initialise the string pointers in the *sval* array to reference user-supplied default strings prior to calling arg_parse. Argtable will only alter the contents of *sval* when matching command line arguments are detected.

REGULAR EXPRESSION COMMAND LINE OPTIONS

commit, update, −−command=commit, −−command=update

```
struct arg_rex
  {
  struct arg_hdr hdr;  /* internal argtable header */
  int count;           /* number of strings returned in sval[] */
  const char **sval;   /* array of pointers to parsed argument strings */
  };
```

Similar to **arg_str** except the string argument values are only accepted if they match a predefined regular expression. Regular expressions are useful for matching command line keywords, particularly if case insensitive strings or pattern matching is required. The regular expression is defined by the *pattern* parameter passed to the *arg_rex* constructor and evaluated using regex. Its behaviour can be controlled via standard regex bit flags. These are passed to argtable via the *flags* parameter in the *arg_rex* constructor. However the only two of the standard regex flags are relevant to argtable, namely REG_EXTENDED (use extended regular expressions rather than basic ones) and REG_ICASE (ignore case). These flags may be logically ORed if desired. See **regex(3)** for more details of regular expression matching.

Restrictions: Argtable does not support **arg_rex** functionality under Microsoft Windows platforms because the Microsoft compilers do include the necessary **regex** support as standard.

FILENAME COMMAND LINE OPTIONS

−o myfile, −Ihome/foo/bar, −−input=~/doc/letter.txt, −−name a.out

struct **arg_file**
```
  {
  struct arg_hdr hdr;      /* internal argtable header */
  int count;               /* number of filename strings returned */
  const char **filename;   /* pointer to full filename string */
  const char **basename;   /* pointer to filename excluding leading path */
  const char **extension;  /* pointer to the filename extension */
  };
```

Similar to **arg_str** but the argument strings are presumed to refer to filenames hence some additional parsing is done to separate out the filename's basename and extension (if they exist). The three arrays filename[], basename[], extension[] each store up to maxcount entries, and the i'th entry of each of these arrays refer to different components of the same string buffer.

For example, **−o /home/heitmann/mydir/foo.txt** would be parsed as:
```
filename[i]  = "/home/heitmann/mydir/foo.txt"
basename[i]  =                      "foo.txt"
extension[i] =                         ".txt"
```

If the filename has no leading path then the basename is the same as the filename. If no extension could be identified then it is given as NULL. Extensions are considered as all text from the last dot in the filename.

Hint: Argtable only ever treats the filenames as strings and never attempts to open them as files or perform any directory lookups on them.

DATE/TIME COMMAND LINE OPTIONS

12/31/04, −d 1982−11−28, −−time 23:59

struct **arg_date**
```
  {
  struct arg_hdr hdr;  /* internal argtable header */
  const char *format;  /* user-supplied date format string that was passed to constructor */
  int count;           /* number of datestamps returned in tmval[] */
  struct tm *tmval;    /* array of datestamps */
  };
```

Accepts a timestamp string from the command line and converts it to *struct tm* format using the system **strptime** function. The time format is defined by the *format* string passed to the *arg_date* constructor, and is passed directly to **strptime**. See **strptime(3)** for more details on the format string.

Restrictions: Argtable does not support **arg_date** functionality under Microsoft Windows because the Microsoft compilers do include the necessary **strptime** support as standard.

REMARK OPTIONS

struct **arg_rem**
```
  {
  struct arg_hdr hdr;  /* internal argtable header */
  };
```

The **arg_rem** struct is a dummy struct in the sense it does not represent a command line option to be parsed. Instead it provides a means to include additional *datatype* and *glossary* strings in the output of the **arg_print_syntax**, **arg_print_syntaxv**, and **arg_print_glossary functions**. As such, **arg_rem** structs may be used in the argument table to insert additional lines of text into the glossary descriptions or to insert additional text fields into the syntax description.

END−OF−TABLE OPTION

```
struct arg_end
  {
  struct arg_hdr hdr;  /* internal argtable header */
  int count;           /* number of errors returned */
  int *error;          /* array of error codes */
  void **parent;       /* pointers to the erroneous command line options */
  const char **argval; /* pointers to the erroneous command line argument values */
  };
```

Every argument table must have an **arg_end** structure as its last entry.
It marks the end of an argument table and stores the error codes generated
by the parser as it processed the argument table.
The *maxerrors* parameter passed to the **arg_end** constructor
specifies the maximum number of errors that the structure can store.
Any further errors are discarded and replaced with the single error code
ARG_ELIMIT which is later reported to the user by the message "too many errors".
A *maxerrors* limit of 20 is quite reasonable.

The **arg_print_errors** function will print the errors stored
in the **arg_end** struct in the same order as they occurred,
so there is no need to understand the internals of the **arg_end** struct.

FUNCTION REFERENCE

int arg_nullcheck (void **argtable)

Returns non−zero if the *argtable[]* array contains any NULL entries up until the terminating **arg_end*** entry. Returns zero otherwise.

int arg_parse (int argc, char **argv, void **argtable)

Parse the command line arguments in *argv[]* using the command line syntax specified in *argtable[]*, returning the number of errors encountered. Error details are recorded in the argument table's **arg_end** structure from where they can be displayed later with the **arg_print_errors** function. Upon a successful parse, the **arg_xxx** structures referenced in *argtable[]* will contain the argument values extracted from the command line.

void arg_print_option (FILE *fp, const char *shortopts, const char *longopts, const char *datatype, const char *suffix)

This function prints an option's syntax, as in **−K|−−scalar=<int>**, where the short options, long options, and datatype are all given as parameters of this function. It is primarily used within the **arg_xxx** structures' *errorfn* functions as a way of displaying an option's syntax inside of error messages. However, it can also be used in user code if desired. The *suffix* string is provided as a convenience for appending newlines and so forth to the end of the display and can be given as NULL if not required.

void arg_print_syntax (FILE *fp, void **argtable, const char *suffix)

Prints the GNU style command line syntax for the given argument table, as in: [−abcv] [−−scalar=<n>] [−o myfile] <file> [<file>]
The *suffix* string is provided as a convenience for appending newlines and so forth to the end of the display and can be given as NULL if not required.

void arg_print_syntaxv (FILE *fp, void **argtable, const char *suffix)

Prints the verbose form of the command line syntax for the given argument table, as in: [−a] [−b] [−c] [−−scalar=<n>] [−o myfile] [−v|−−verbose] <file> [<file>]
The *suffix* string is provided as a convenience for appending newlines and so forth to the end of the display and can be given as NULL if not required.

void arg_print_glossary (FILE *fp, void **argtable, const char *format)

Prints a glossary table describing each option in the given argument table. The *format* string is passed to printf to control the formatting of each entry in the the glossary. It must have exactly two "%s" format parameters as in "%−25s %s\n", the first is for the option's syntax and the second for its glossary string. If an option's glossary string is NULL then that option in omitted from the glossary display.

void arg_print_glossary_gnu (FILE *fp, void **argtable)

An alternate form of **arg_print_glossary()** that prints the glossary using strict GNU formatting conventions wherein long options are vertically aligned in a second column, and lines are wrapped at 80 characters.

void arg_print_errors (FILE *fp, struct arg_end *end, const char *progname)

Prints the details of all errors stored in the *end* data structure. The *progname* string is prepended to each error message.

void arg_freetable (void ** argtable, size_t n)

Deallocates the memory used by each **arg_xxx** struct referenced by *argtable[]*. It does this by calling **free** for each of the *n* pointers in the argtable array and then nulling them for safety.

FILES

/usr/local/include/argtable2.h
/usr/local/lib/libargtable2.a
/usr/local/lib/libargtable2.so
/usr/local/man3/argtable2.3
/usr/local/share/doc/argtable2/
/usr/local/share/doc/argtable2/example/

AUTHOR

Stewart Heitmann <sheitmann@users.sourceforge.net>