NAME

clig – generate a command line parser and/or basic manual page

SYNOPSIS

clig [-t types] [-o outprefix] [-m manFile] [-mx manExt] [-d] infile

OPTIONS

-t
: the types of output to generate. Currently supported types are 'C', 'tcl' and 'man',
1 or more String values
Default: 'C' 'man'

-o
: name of .c and .h file if type C is requested Default: infile with suffix removed,
1 String value

-m
: manual page to edit or generate if type man is requested Default: 'Name' specified in input file,
1 String value

-mx
: extension of manual page,
1 String value
Default: '.1'

-d
: generate the function showOptionValues if type C is requested .

infile
: name of file which contains the command line specification.

DESCRIPTION

Clig reads *infile* and creates the output as requested by option **-t**. Most often you will just use the default to create a command line interpreter for your C program as well as the skeleton of a manual page. One of the main reasons to use **clig** is, that besides the command line interpreter, a **usage()** function and a manual page are generated, which are always up-to-date with respect to the options actually understood by the program.

Currently, options of type **Flag**, **String**, **Int**, **Long**, **Float** and **Double** are supported. Except for **Flag** options, all option-descriptions allow to specify how many arguments are expected for the option and whether the option is mandatory or not. For non-mandatory options, default values can be specified and will be returned by the generated command line interpreter if necessary. In addition, for floating point and integer options, a range can be declared, and the interpreter will exit the program with an error message, if the option's argument lies outside the given range.

Clig is implemented in Tcl and *infile* actually contains tcl-code, but under normal circumstances the user does not need to know anything about tcl, because the syntax of the description-file is described below in section "DESCRIPTION FILE" and in the manual pages **clig_*** as noted in the section "SEE ALSO". (I admit that this is probably a lie, because under 'normal circumstances' *infile* may contain syntax errors resulting in really tcl-ish error messages.)

DESCRIPTION FILE

The description-file, *infile*, is a line-oriented ascii-file. Each line contains a command which either describes an option or specifies additional information necessary to generate the command line interpreter or the manual page. A command starts with the command-name followed by mandatory parameters and possibly followed by options. If a command does not fit on a line, it may be continued on the next line if the previous line is terminated with a backslash (\).

The commands are described in detail in their own manual pages, but let us consider a simple example, e.g. a command to describe an option named **-fs** which wants exactly one parameter of type **float** and will be used in your program as a font size. (Note the backslash on the next line!)

**Float -fs fontsize **
{size of font in points} -d 11.0 -r 8.0 19.5

The command's name is **Float**. It has three fixed parameters: The first, **-fs**, is the string to be detected by the command line interpreter. The second, **fontsize**, is the name of a variable your program will use to receive the value found on the command line. The third parameter, i.e. everything between the braces, is a string which will be printed by **usage()**. The **-d** (for default) specifies that **fontsize** will be set to the default of 11.0, if **-fs** is not found on the command line. With **-r** (for range) you make sure that the command line interpreter will refuse any values for **-fs** which are not within the given range.

Mandatory Commands

The description file *infile* must contain the commands **Name** and **Usage**. The first gives a name to your program and the second declares a short (one-line) description for it. Both pieces of information will be used in the **usage()**-message as well as in the manual page.

Other Commands

Please read the manual pages listed under "SEE ALSO" below which start with **clig_** to find a detailed description of all commands available in the description file *infile*. An example-*infile* can be found as |Example|/cmdline.cli.

OUTPUT TYPE C

Clig generates the files *outprefix*.c and *outprefix*.h, where *outprefix*.h contains all declarations necessary to use the services defined in the *outprefix*.c. If a prefix is not given on the command line, it defaults to the name of the input file with any suffix removed.

parseCmdline

The output files implement a command line interpreter with the prototype

Cmdline *parseCmdline(int argc, char **argv);

which is meant to be called by your **main()** to interpret the command line given as **argc** and **argv**.

usage

The output files also implement the function

void usage(void);

custom made from the information in *infile*. IF **parseCmdline()** encounters an undeclared option on the command line, it immediately calls **usage()**. Therefore you should normally declare neither -h nor --help as options for your program so that usage-message can be requested with any of these options. Normally it is not necessary to call **usage()** directly, but it is possible.

Cmdline

The type **Cmdline**, a pointer to which is returned by **parseCmdline()**, is also declared in the generated header file. It is custom-made according to what is found in *infile*. If, for example, *infile* contains the declaration

```
Float -pi pi {your personal approximation of pi}
```

the structure will have the three slots **piP**, **pi** and **piC**, where **piP** is a boolean set to 1 if and only if the option **-r** is found on the command line. In that case **piC** contains the number of parameters found for **-r**. The parameters found are stored in **pi** itself. It is either of type **float*** or **float** depending on whether option **-r** may at all have more than one argument. In the example above, this is not the case, but if you declare

```
Float -ival ival {interval to consider} -c 2 2
```

in *infile*, **ival** must store 2 values and consequently it will be of type **float*** with sufficient memory allocated.

Program

To be conveniently available for error message and for use in **usage()**, **parseCmdline()** will also set the global variable

char *Program;

to the tail of **argv[0]**.

showOptionValues

If so requested with option **-d**, the output will contain a function

void showOptionValues(void);

which can be called after **parseCmdline()** to print the structure **Cmdline** to stdout. This is merely for debugging purposes.

Example Main Program

The following example demonstrates the use of the command line parser, provided that *outprefix* = **cmdline** and that **clig** was called with option **-d** to generate the function **showOptionValues()**.

```
#include "cmdline.h"

int main(int argc, char **argv)
{
  Cmdline *cmd = parseCmdline(argc, argv);
  showOptionValues();

  /* Program is set by parseCmdline */
  printf("Program = ‘%s’\n", Program);

  /*****
    Your code goes here. Option parameters and cleaned-up
    argc and argv are referenced with cmd->...
  *****/

  return 0;
}
```

OUTPUT TYPE tcl

In fact, output type **tcl** is not yet supported. However this does not mean you cannot use **::clig** to instrument your Tcl-scripts. It only means that you must have **::clig** installed on the machine were an instrumented script should run. Read **clig_parseCmdline(n)** to learn how to instrument your Tcl-script with **::clig**.

OUTPUT TYPE man

Clig can generate or edit a manual page in *roff format. The name of the manual page can be specified with option **-m** and its suffix can be specified with **-mx**. By default, the name will be the string specified with the **Name**-command in the description file and the default suffix is **.1**.

If the manual page file exists, clig edits specially marked sections of the file by filling in up-to-date information. If the file does not yet exist, it copies a template file into it and then edits the template in the same way.

Clig is able to fill out the manual page sections 'NAME', 'SYNOPSIS', 'OPTIONS' and, in a very limited way, 'DESCRIPTION'. In addition it will supply a default title macro (.TH). The lines of the manual page file which are replaced by clig must be clearly marked by matching pairs of tag lines like

```
.\" cligPart <section>
```

and

```
.\" cligPart <section> end
```

where <section> is one of the section names listed above.

The idea is, that you edit the manual page, while clig fills in the parts that can be deduced from the description file. If you don't like what clig fills in, simply remove both tag-lines of a section and clig will leave it alone. You certainly want to do this for the DESCRIPTION-section.

SEE ALSO

|SEEALSO|

NAME

::clig::Flag – declare a flag (boolean, classic) option

SYNOPSIS

package require clig
namespace import ::clig::*
setSpec *db*

Flag -*opt varname usage*

DESCRIPTION

The **Flag** command declares an option without any parameters. The command line parser merely checks for the existence of *opt* on the command line.

tcl

If **parseCmdline** finds *opt* on the command line, variable *varname* in the caller’s context is set to 1. If *opt* is not on the command line, the variable is not set.

C

If *opt* is found on the command line the slot *variable***P** in the structure returned by the parser will be set to a non-zero value, otherwise it will be 0.

Example use of **Flag**:

```
Flag -v verbose {verbose operation}
```

SEE ALSO

|SEEALSO|

NAME

::clig::parseCmdline – command line interpreter for Tcl

SYNOPSIS

package require clig
namespace import ::clig::*

setSpec *var*
parseCmdline *_spec argv0 argc argv*

DESCRIPTION

This manual page describes how to instrument your Tcl-scripts with a command line parser. It requires that package clig is installed whenever your script is run. To find out how to create a parser which is independent of clig, read clig(1). (Well, don't, it is not yet implemented.)

The options to be understood by your script must be declared with calls to ::clig::Flag, ::clig:String etc., which is best done in a separate file, e.g. cmdline.cli, to be sourced by your script. Having these declarations in a separate file allows you to run **clig**, the program, on that file to create a basic manual page.

setSpec

The option-declaring functions want to store the declarations in an array with a globally accessible name. For compatibility with older software, the name of this array can not be passed as a parameter. Consequently, your script must declare it before sourcing **cmdline.cli**.

A call like

```
setSpec ::main
```

declares **::main** as the database (array) to be filled by subsequent calls to ::clig::Flag and the like. The array should not contain anything except entries created with the declarator functions.

parseCmdline

After declaring the database and sourcing your declarations from **cmdline.cli** your script is ready to call parseCmdline. This is typically done with:

```
set Program [file tail $argv0]
if {[catch {parseCmdline ::main $Program $argc $argv} err]} {
  puts stderr $err
  exit 1
}
```

If **parseCmdline** finds an unknown option in **$argv**, it prints a usage-message to stderr and exits the script with exit-code 1. If it finds any other errors, like numerical arguments being out of range, it prints an appropriate error-message to stderr and also exits your script with code 1. Setting **Program** and passing it to **parseCmdline** instead of **$argv0** results in nicer error-messages.

If no errors occur, **parseCmdline** enters the values found into variables of its callers context. The names of these variables are those declared with the declarator functions. For example with a declaration like

```
Float -ival ival {interval to take into account} -c 2 2
```

the caller will find the variable **ival** set to a list with 2 elements, if option **-ival** was found with two numeric arguments in **$argv**.

If option **-ival** is not existent on the given command line, **ival** will not be set. Consequently, it is best to not set the declared variables to any value before calling **parseCmdline**.

Summary

The typical skeleton of your script should look like

```
package require clig
namespace import ::clig::*

setSpec ::main
source [file join path to your installed base cmdline.cli]
set Program [file tail $argv0]
if {[catch {parseCmdline ::main $Program $argc $argv} err]} {
  puts stderr $err
  exit 1
}
```

REMARK

Of course **parseCmdline** can be called from a within any **proc** with a specification database previously filled with the declarator functions. I am not using **OptProc** because it is not documented and the declaration syntax used here was used for C-programs probably long before it existed.

SEE ALSO

|SEEALSO|

NAME

::clig::Description – set long description text to be included in a manual page

SYNOPSIS

package require clig
namespace import ::clig::*
setSpec *db*

Description *text*

DESCRIPTION

The **Description** command should not be used. Instead, the respective section in the generated manual page should be filled out. The main reason for this advice is, that clig copies *description* AS IS into the manual page, so it can be typed into the manual page file in the first place. Please remember that the generated manual page needs some hand-tuning anyway, because for example the 'SEE ALSO'-section cannot be generated. Since this command should not be used, no example is given.

SEE ALSO

|SEEALSO|

NAME

::clig::Float – declare an option with parameters of type float

SYNOPSIS

package require clig
namespace import ::clig::*
setSpec *db*

Float -*opt varname usage* [**-c** *min max*] {[**-d** *default ...*] | [**-m**]} [**-r** *rmin rmax*]

DESCRIPTION

The **Float** command declares *-opt* to have zero or more floating point arguments. The parameters *variable* and *usage* as well as the options **-c**, **-d** and **-m** are described in **clig::String**(n).

-r instructs the parser to check that arguments given to *-opt* are within the inclusive range between *rmin* and *rmax*. Special values for *rmin* and *rmax* are **-oo** and **oo"** denoting negative and positive infinity thereby not constraining the parameters of **-opt**.

WARNING: The implementation of option **-d** is currently incomplete insofar, as the specified default values are not checked by **clig**, i.e. the following would go unnoticed:

-d 1.0 -r 500.0 600.0

In this case, if *option* is not on the command line, the clig parser will set variable *varname* for its caller to the out-of-range value 1.0. (Maybe this is a feature?)

Example use of **Float**:

```
Float -p prob {list of probabilities adding up to one} \
  -r 0.0 1.0  -d 0.1 0.2 0.7 \
  -c 1 oo
```

Please note, that the additional constraint requiring the values summing to 1.0 cannot be checked by the parser.

PARSER RESULT

tcl

Within a Tcl-script, the parser will set the variable with name *varname* in the stack-frame of its caller if and only if option *-opt* is found on the command line or has a default value.

C

The structure returned by the parser contains the slots *varname***P**, *varname* and *varname***C**. The slot *varname***P** will be set to a non-zero value, if and only if the option *-opt* is found on the command line or has a default value. Otherwise it will be 0. The slot *varname***C** will be set to the number of arguments found for *-opt*. If the number of arguments for *-opt* is allowed to be greater than 1, slot *varname* has type **float***, otherwise it has type **float**.

SEE ALSO

|SEEALSO|

NAME

::clig::Int – declare an option with parameters of type int

SYNOPSIS

package require clig
namespace import ::clig::*
setSpec *db*

Int -*opt varname usage* [**-c** *min max*] {[**-d** *default ...*] | [**-m**]} [**-r** *rmin rmax*]

DESCRIPTION

The **Int** command declares *-opt* to have zero or more integer arguments. The parameters *variable* and *usage* as well as the options **-c**, **-d** and **-m** are described in **clig::String**(n).

-r instructs the parser to check that arguments given to *-opt* are within the inclusive range between *rmin* and *rmax*. Special values for *rmin* and *rmax* are **-oo** and **oo"** denoting negative and positive infinity thereby not constraining the parameters of **-opt**.

WARNING: The implementation of option **-d** is currently incomplete insofar, as the specified default values are not checked by **clig**, i.e. the following would go unnoticed:

-d 1.0 -r 500 600

In this case, if *option* is not on the command line, the clig parser will set variable *varname* for its caller to the out-of-range value 1.0. (Maybe this is a feature?)

Example use of **Int**:

```
Int -ind indices {list of indices to use} \
   -r 0 99  \
   -d 0 10 20 30 40 50 60 70 80 90 \
   -c 1 oo
```

PARSER RESULT

tcl

Within a Tcl-script, the parser will set the variable with name *varname* in the stack-frame of its caller if and only if option *-opt* is found on the command line or has a default value.

C

The structure returned by the parser contains the slots *varname***P**, *varname* and *varname***C**. The slot *varname***P** will be set to a non-zero value, if and only if the option *-opt* is found on the command line or has a default value. Otherwise it will be 0. The slot *varname***C** will be set to the number of arguments found for *-opt*. If the number of arguments for *-opt* is allowed to be greater than 1, slot *varname* has type **int***, otherwise it has type **int**.

SEE ALSO

|SEEALSO|

NAME

::clig::Name – set the program name to be used in the manual page

SYNOPSIS

package require clig
namespace import ::clig::*
setSpec *db*

Name *program-name*

DESCRIPTION

The **Name**-command specifies the name of the program. The string *program-name* is used in the generated manual page and also specifies the default name of the manual page. However the specified name is not used anywhere else in the parser because use of argv[0] is preferred there.

SEE ALSO

|SEEALSO|

NAME

::clig::Long – declare an option with parameters of type long

SYNOPSIS

package require clig
namespace import ::clig::*
setSpec *db*

Long -*opt varname usage* [**-c** *min max*] {[**-d** *default ...*] | [**-m**]} [**-r** *rmin rmax*]

DESCRIPTION

The **Long** command works exactly as the **Int** command except that the generated C parser works with values of type **long** instead of **int**. For the Tcl parser it does not make a difference.

SEE ALSO

|SEEALSO|

NAME

::clig::Double – declare an option with parameters of type double

SYNOPSIS

package require clig
namespace import ::clig::*
setSpec *db*

Double -*opt varname usage* [**-c** *min max*] {[**-d** *default ...*] | [**-m**]} [**-r** *rmin rmax*]

DESCRIPTION

The **Double** command works exactly as the **Float** command except that the generated C parser works with values of type **double** instead of **float**. For the Tcl parser it does not make a difference.

SEE ALSO

|SEEALSO|

NAME

::clig::Rest – declare command line arguments not associated with any option

SYNOPSIS

package require clig
namespace import ::clig::*
setSpec *db*

Rest *name usage* [**-c** *lo hi*]

DESCRIPTION

The **Rest** command specifies, what the parser should do with command line arguments not associated with any option. Option **-c** specifies how man non-option arguments are allowed on the command line (see clig::String(n)). The default values for *lo* and *hi* are 0 and **oo** respectively.

Without a **Rest**-declaration the parser will generate an error message and terminate the program.

If a **Rest** is declared, non-option arguments are returned by the parser as described below.

PARSER RESULT

tcl

Within a Tcl-script, the parser will set the variable with name *varname* in the stack-frame of its caller to the arguments on the command line which are not parameters of any option.

C

Mainly for historical reasons, the structure returned by the parser contains the slots **int argc** and **char **argv**. They contain the arguments on the command line which are not parameters of any option.

Example use of **Rest**:

```
Rest infiles {input files} -c 1 oo
```

SEE ALSO

|SEEALSO|

NAME

::clig::String – declare an option with parameters of type string

SYNOPSIS

package require clig
namespace import ::clig::*
setSpec *db*

String -*opt varname usage* [**-c** *min max*] {[**-d** *default ...*] | [**-m**]}

DESCRIPTION

The **String** command declares *-opt* to have zero or more string arguments.

varname
: declares that arguments found for *-opt* are passed to the caller of the parser in a variable (tcl) or slot (C) of that name.

usage
: is a short text describing *-opt*. It is used in the generated manual or in a usage-message printed by the parser if necessary.

-c
: specifies the minimum and maximum number of parameters *-opt* may have. If less than *min* or more than *max* are found, the parser prints an error message to stderr and terminates the calling process. Use *min==max* to request exactly so many arguments. As a special value for **max**, **oo** is understood as positive infinity.

-d
: stores a (list of) default value(s) for option *-opt*. The parser pretends these were found on the command line, if option *-opt* is not given.
WARNING: The number of default values is not checked against the minimum and maximum values given for **-c**.

-m
: declares that *-opt* is mandatory. Put another way, the “option” is not really optional but must show up on the command line. If it does not, the parser prints an error message to stderr and exits the calling process. This can not be used together with **-d**.

PARSER RESULT

tcl

Within a Tcl-script, the parser will set the variable with name *varname* in the stack-frame of its caller if and only if option *-opt* is found on the command line or has a default value.

C

The structure returned by the parser contains the slots *varname***P**, *varname* and *varname***C**. The slot *varname***P** will be set to a non-zero value, if and only if the option *-opt* is found on the command line or has a default value. Otherwise it will be 0. The slot *varname***C** will be set to the number of arguments found for *-opt*. If the number of arguments for *-opt* is allowed to be greater than 1, slot *varname* has type **char****, otherwise it has type **char***.

SEE ALSO

|SEEALSO|

NAME

::clig::Usage – declare single line usage-string

SYNOPSIS

package require clig
namespace import ::clig::*
setSpec *db*

Usage *one-liner*

DESCRIPTION

The **Usage** command specifies a one line description of the program's intent and function. The string *one-liner* is printed --- besides other sutff --- if the parser encounters an unknown option. Remember to enclose *one-liner* in braces. A (not so) typical example would be

```
Usage {program to delete your home directory}
```

In fact, the *one-liner* can be longer than one line. Best results are obtained, if *one-liner* is not put in braces but is surrounded by double quotes. The quoted string may than contain newlines. Clig automatically adds backslashes where necessary in the generated C-code so that the C-compiler does not go wild.

SEE ALSO

|SEEALSO|

NAME

::clig::Version – declare version

SYNOPSIS

package require clig
namespace import ::clig::*
setSpec *db*

Version *version*

DESCRIPTION

The **Version** command specifies a string to be used as a version-identification of the generated parser. Currently it is merely printed as part of the usage-information. An example using my favorite type of version numbers would be

```
Version 2000-01-31
```

SEE ALSO

|SEEALSO|

NAME

::clig::Commandline – declare variable to store concatenated argv

SYNOPSIS

package require clig
namespace import ::clig::*
setSpec *db*

Commandline *varname*

DESCRIPTION

The command **Commandline** records in the database *db* specified in the most recent call to **setSpec** the name *varname* of a variable to be set the concatenated elements of the command line by a clig-parser.

Example

With a declaration like

```
Commandline tool
```

a C-program calling

```
Cmdline *cmd = parseCmdline(argc, argv);
```

will find in **cmd->tool** all elements of argv joined into one string with intervening blanks.

BUG

This even works with the command line parser for Tcl, **::clig::parseCmdline**, although in contrast to C it is rather useless in Tcl.

SEE ALSO

|SEEALSO|