| LCOV - code coverage report |
| --- |
|  |
| | Current view: | [top level](http://docs.google.com/index.html) - [Common/include](http://docs.google.com/index.html) - SimpleOpt.h (source / [functions](http://docs.google.com/SimpleOpt.h.func.html)) |  |  | Hit | Total | Coverage | | --- | --- | --- | --- | --- | --- | --- | | Test: | EBM9K Converter Coverage Report |  | Lines: | 146 | 207 | 70.5 % | | Date: | 2012-03-23 |  | Functions: | 9 | 10 | 90.0 % | | Legend: | Lines: hit not hit | Branches: + taken - not taken # not executed |  | Branches: | 103 | 202 | 51.0 % | |  | |
|  |

|  |
| --- |
| Branch data Line data Source code  1 : : /\*! @file SimpleOpt.h  2 : :   3 : : @version 3.2  4 : :   5 : : @brief A cross-platform command line library which can parse almost any  6 : : of the standard command line formats in use today. It is designed   7 : : explicitly to be portable to any platform and has been tested on Windows   8 : : and Linux. See CSimpleOptTempl for the class definition.  9 : :   10 : : @section features FEATURES  11 : :   12 : : - MIT Licence allows free use in all software (including GPL   13 : : and commercial)  14 : : - multi-platform (Windows 95/98/ME/NT/2K/XP, Linux, Unix)  15 : : - supports all lengths of option names:  16 : : <table width="60%">  17 : : <tr><td width="30%"> -   18 : : <td>switch character only (e.g. use stdin for input)  19 : : <tr><td> -o   20 : : <td>short (single character)  21 : : <tr><td> -long   22 : : <td>long (multiple character, single switch character)  23 : : <tr><td> --longer   24 : : <td>long (multiple character, multiple switch characters)  25 : : </table>  26 : : - supports all types of arguments for options:  27 : : <table width="60%">  28 : : <tr><td width="30%"> --option   29 : : <td>short/long option flag (no argument)  30 : : <tr><td> --option ARG   31 : : <td>short/long option with separate required argument  32 : : <tr><td> --option=ARG   33 : : <td>short/long option with combined required argument  34 : : <tr><td> --option[=ARG]   35 : : <td>short/long option with combined optional argument  36 : : <tr><td> -oARG   37 : : <td>short option with combined required argument  38 : : <tr><td> -o[ARG]   39 : : <td>short option with combined optional argument  40 : : </table>  41 : : - supports options with multiple or variable numbers of arguments:  42 : : <table width="60%">  43 : : <tr><td width="30%"> --multi ARG1 ARG2   44 : : <td>Multiple arguments  45 : : <tr><td> --multi N ARG-1 ARG-2 ... ARG-N   46 : : <td>Variable number of arguments  47 : : </table>  48 : : - supports case-insensitive option matching on short, long and/or   49 : : word arguments.  50 : : - supports options which do not use a switch character. i.e. a special   51 : : word which is construed as an option.   52 : : e.g. "foo.exe open /directory/file.txt"   53 : : - supports clumping of multiple short options (no arguments) in a string   54 : : e.g. "foo.exe -abcdef file1" <==> "foo.exe -a -b -c -d -e -f file1"  55 : : - automatic recognition of a single slash as equivalent to a single   56 : : hyphen on Windows, e.g. "/f FILE" is equivalent to "-f FILE".  57 : : - file arguments can appear anywhere in the argument list:  58 : : "foo.exe file1.txt -a ARG file2.txt --flag file3.txt file4.txt"  59 : : files will be returned to the application in the same order they were   60 : : supplied on the command line  61 : : - short-circuit option matching: "--man" will match "--mandate"  62 : : invalid options can be handled while continuing to parse the command   63 : : line valid options list can be changed dynamically during command line  64 : : processing, i.e. accept different options depending on an option   65 : : supplied earlier in the command line.  66 : : - implemented with only a single C++ header file  67 : : - optionally use no C runtime or OS functions  68 : : - char, wchar\_t and Windows TCHAR in the same program  69 : : - complete working examples included  70 : : - compiles cleanly at warning level 4 (Windows/VC.NET 2003), warning   71 : : level 3 (Windows/VC6) and -Wall (Linux/gcc)  72 : :   73 : : @section usage USAGE  74 : :   75 : : The SimpleOpt class is used by following these steps:  76 : :   77 : : <ol>  78 : : <li> Include the SimpleOpt.h header file  79 : :   80 : : <pre>  81 : : \#include "SimpleOpt.h"  82 : : </pre>  83 : :   84 : : <li> Define an array of valid options for your program.  85 : :   86 : : <pre>  87 : : @link CSimpleOptTempl::SOption CSimpleOpt::SOption @endlink g\_rgOptions[] = {  88 : : { OPT\_FLAG, \_T("-a"), SO\_NONE }, // "-a"  89 : : { OPT\_FLAG, \_T("-b"), SO\_NONE }, // "-b"  90 : : { OPT\_ARG, \_T("-f"), SO\_REQ\_SEP }, // "-f ARG"  91 : : { OPT\_HELP, \_T("-?"), SO\_NONE }, // "-?"  92 : : { OPT\_HELP, \_T("--help"), SO\_NONE }, // "--help"  93 : : SO\_END\_OF\_OPTIONS // END  94 : : };  95 : : </pre>  96 : :   97 : : Note that all options must start with a hyphen even if the slash will  98 : : be accepted. This is because the slash character is automatically  99 : : converted into a hyphen to test against the list of options.   100 : : For example, the following line matches both "-?" and "/?"   101 : : (on Windows).  102 : :   103 : : <pre>  104 : : { OPT\_HELP, \_T("-?"), SO\_NONE }, // "-?"  105 : : </pre>  106 : :   107 : : <li> Instantiate a CSimpleOpt object supplying argc, argv and the option   108 : : table  109 : :   110 : : <pre>  111 : : @link CSimpleOptTempl CSimpleOpt @endlink args(argc, argv, g\_rgOptions);  112 : : </pre>  113 : :   114 : : <li> Process the arguments by calling Next() until it returns false.   115 : : On each call, first check for an error by calling LastError(), then   116 : : either handle the error or process the argument.  117 : :   118 : : <pre>  119 : : while (args.Next()) {  120 : : if (args.LastError() == SO\_SUCCESS) {  121 : : handle option: use OptionId(), OptionText() and OptionArg()  122 : : }  123 : : else {  124 : : handle error: see ESOError enums  125 : : }  126 : : }  127 : : </pre>  128 : :   129 : : <li> Process all non-option arguments with File(), Files() and FileCount()  130 : :   131 : : <pre>  132 : : ShowFiles(args.FileCount(), args.Files());  133 : : </pre>  134 : :   135 : : </ol>  136 : :   137 : : @section notes NOTES  138 : :   139 : : - In MBCS mode, this library is guaranteed to work correctly only when  140 : : all option names use only ASCII characters.  141 : : - Note that if case-insensitive matching is being used then the first  142 : : matching option in the argument list will be returned.  143 : :   144 : : @section licence MIT LICENCE  145 : :   146 : : The licence text below is the boilerplate "MIT Licence" used from:  147 : : http://www.opensource.org/licenses/mit-license.php  148 : :   149 : : Copyright (c) 2006-2007, Brodie Thiesfield  150 : :   151 : : Permission is hereby granted, free of charge, to any person obtaining a  152 : : copy of this software and associated documentation files (the "Software"),  153 : : to deal in the Software without restriction, including without limitation  154 : : the rights to use, copy, modify, merge, publish, distribute, sublicense,  155 : : and/or sell copies of the Software, and to permit persons to whom the  156 : : Software is furnished to do so, subject to the following conditions:  157 : :   158 : : The above copyright notice and this permission notice shall be included  159 : : in all copies or substantial portions of the Software.  160 : :   161 : : THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS   162 : : OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF   163 : : MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.   164 : : IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY   165 : : CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,   166 : : TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE   167 : : SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.  168 : : \*/  169 : :   170 : : /\*! @mainpage  171 : :   172 : : <table>  173 : : <tr><th>Library <td>SimpleOpt  174 : : <tr><th>Author <td>Brodie Thiesfield [code at jellycan dot com]  175 : : <tr><th>Source <td>http://code.jellycan.com/simpleopt/  176 : : </table>  177 : :   178 : : @section SimpleOpt SimpleOpt  179 : :   180 : : A cross-platform library providing a simple method to parse almost any of  181 : : the standard command-line formats in use today.  182 : :   183 : : See the @link SimpleOpt.h SimpleOpt @endlink documentation for full   184 : : details.  185 : :   186 : : @section SimpleGlob SimpleGlob  187 : :   188 : : A cross-platform file globbing library providing the ability to  189 : : expand wildcards in command-line arguments to a list of all matching   190 : : files.  191 : :   192 : : See the @link SimpleGlob.h SimpleGlob @endlink documentation for full   193 : : details.  194 : : \*/  195 : :   196 : : #ifndef INCLUDED\_SimpleOpt  197 : : #define INCLUDED\_SimpleOpt  198 : :   199 : : // Default the max arguments to a fixed value. If you want to be able to   200 : : // handle any number of arguments, then predefine this to 0 and it will   201 : : // use an internal dynamically allocated buffer instead.  202 : : #ifdef SO\_MAX\_ARGS  203 : : # define SO\_STATICBUF SO\_MAX\_ARGS  204 : : #else  205 : : # include <stdlib.h> // malloc, free  206 : : # include <string.h> // memcpy  207 : : # define SO\_STATICBUF 50  208 : : #endif  209 : :   210 : : //! Error values  211 : : typedef enum \_ESOError  212 : : {  213 : : //! No error  214 : : SO\_SUCCESS = 0,   215 : :   216 : : /\*! It looks like an option (it starts with a switch character), but   217 : : it isn't registered in the option table. \*/  218 : : SO\_OPT\_INVALID = -1,   219 : :   220 : : /\*! Multiple options matched the supplied option text.   221 : : Only returned when NOT using SO\_O\_EXACT. \*/  222 : : SO\_OPT\_MULTIPLE = -2,   223 : :   224 : : /\*! Option doesn't take an argument, but a combined argument was   225 : : supplied. \*/  226 : : SO\_ARG\_INVALID = -3,   227 : :   228 : : /\*! SO\_REQ\_CMB style-argument was supplied to a SO\_REQ\_SEP option  229 : : Only returned when using SO\_O\_PEDANTIC. \*/  230 : : SO\_ARG\_INVALID\_TYPE = -4,   231 : :   232 : : //! Required argument was not supplied  233 : : SO\_ARG\_MISSING = -5,   234 : :   235 : : /\*! Option argument looks like another option.   236 : : Only returned when NOT using SO\_O\_NOERR. \*/  237 : : SO\_ARG\_INVALID\_DATA = -6   238 : : } ESOError;  239 : :   240 : : //! Option flags  241 : : enum \_ESOFlags  242 : : {  243 : : /\*! Disallow partial matching of option names \*/  244 : : SO\_O\_EXACT = 0x0001,   245 : :   246 : : /\*! Disallow use of slash as an option marker on Windows.   247 : : Un\*x only ever recognizes a hyphen. \*/  248 : : SO\_O\_NOSLASH = 0x0002,   249 : :   250 : : /\*! Permit arguments on single letter options with no equals sign.   251 : : e.g. -oARG or -o[ARG] \*/  252 : : SO\_O\_SHORTARG = 0x0004,   253 : :   254 : : /\*! Permit single character options to be clumped into a single   255 : : option string. e.g. "-a -b -c" <==> "-abc" \*/  256 : : SO\_O\_CLUMP = 0x0008,   257 : :   258 : : /\*! Process the entire argv array for options, including the   259 : : argv[0] entry. \*/  260 : : SO\_O\_USEALL = 0x0010,   261 : :   262 : : /\*! Do not generate an error for invalid options. errors for missing   263 : : arguments will still be generated. invalid options will be   264 : : treated as files. invalid options in clumps will be silently   265 : : ignored. \*/  266 : : SO\_O\_NOERR = 0x0020,   267 : :   268 : : /\*! Validate argument type pedantically. Return an error when a   269 : : separated argument "-opt arg" is supplied by the user as a   270 : : combined argument "-opt=arg". By default this is not considered   271 : : an error. \*/  272 : : SO\_O\_PEDANTIC = 0x0040,   273 : :   274 : : /\*! Case-insensitive comparisons for short arguments \*/  275 : : SO\_O\_ICASE\_SHORT = 0x0100,   276 : :   277 : : /\*! Case-insensitive comparisons for long arguments \*/  278 : : SO\_O\_ICASE\_LONG = 0x0200,   279 : :   280 : : /\*! Case-insensitive comparisons for word arguments   281 : : i.e. arguments without any hyphens at the start. \*/  282 : : SO\_O\_ICASE\_WORD = 0x0400,   283 : :   284 : : /\*! Case-insensitive comparisons for all arg types \*/  285 : : SO\_O\_ICASE = 0x0700   286 : : };  287 : :   288 : : /\*! Types of arguments that options may have. Note that some of the \_ESOFlags  289 : : are not compatible with all argument types. SO\_O\_SHORTARG requires that  290 : : relevant options use either SO\_REQ\_CMB or SO\_OPT. SO\_O\_CLUMP requires   291 : : that relevant options use only SO\_NONE.  292 : : \*/  293 : : typedef enum \_ESOArgType {  294 : : /\*! No argument. Just the option flags.  295 : : e.g. -o --opt \*/  296 : : SO\_NONE,   297 : :   298 : : /\*! Required separate argument.   299 : : e.g. -o ARG --opt ARG \*/  300 : : SO\_REQ\_SEP,   301 : :   302 : : /\*! Required combined argument.   303 : : e.g. -oARG -o=ARG --opt=ARG \*/  304 : : SO\_REQ\_CMB,   305 : :   306 : : /\*! Optional combined argument.   307 : : e.g. -o[ARG] -o[=ARG] --opt[=ARG] \*/  308 : : SO\_OPT,   309 : :   310 : : /\*! Multiple separate arguments. The actual number of arguments is  311 : : determined programatically at the time the argument is processed.  312 : : e.g. -o N ARG1 ARG2 ... ARGN --opt N ARG1 ARG2 ... ARGN \*/  313 : : SO\_MULTI  314 : : } ESOArgType;  315 : :   316 : : //! this option definition must be the last entry in the table  317 : : #define SO\_END\_OF\_OPTIONS { -1, NULL, SO\_NONE }  318 : :   319 : : #ifdef \_DEBUG  320 : : # ifdef \_MSC\_VER  321 : : # include <crtdbg.h>  322 : : # define SO\_ASSERT(b) \_ASSERTE(b)  323 : : # else  324 : : # include <assert.h>  325 : : # define SO\_ASSERT(b) assert(b)  326 : : # endif  327 : : #else  328 : : # define SO\_ASSERT(b) //!< assertion used to test input data  329 : : #endif  330 : :   331 : : // ---------------------------------------------------------------------------  332 : : // MAIN TEMPLATE CLASS  333 : : // ---------------------------------------------------------------------------  334 : :   335 : : /\*! @brief Implementation of the SimpleOpt class \*/  336 : : template<class SOCHAR>  337 : : class CSimpleOptTempl  338 : : {  339 : : public:  340 : : /\*! @brief Structure used to define all known options. \*/  341 : : struct SOption {  342 : : /\*! ID to return for this flag. Optional but must be >= 0 \*/  343 : : int nId;   344 : :   345 : : /\*! arg string to search for, e.g. "open", "-", "-f", "--file"   346 : : Note that on Windows the slash option marker will be converted  347 : : to a hyphen so that "-f" will also match "/f". \*/  348 : : const SOCHAR \* pszArg;  349 : :   350 : : /\*! type of argument accepted by this option \*/  351 : : ESOArgType nArgType;   352 : : };  353 : :   354 : : /\*! @brief Initialize the class. Init() must be called later. \*/  355 : : CSimpleOptTempl()   356 : : : m\_rgShuffleBuf(NULL)   357 : : {   358 : : Init(0, NULL, NULL, 0);   359 : : }  360 : :   361 : : /\*! @brief Initialize the class in preparation for use. \*/  362 : : CSimpleOptTempl(  363 : : int argc,   364 : : SOCHAR \* argv[],   365 : : const SOption \* a\_rgOptions,   366 : : int a\_nFlags = 0  367 : : )   368 : 3339 : : m\_rgShuffleBuf(NULL)   369 : 3339 : {   370 : 3339 : Init(argc, argv, a\_rgOptions, a\_nFlags);   371 : : }  372 : :   373 : : #ifndef SO\_MAX\_ARGS  374 : : /\*! @brief Deallocate any allocated memory. \*/  375 [ - + ][ - + ]: 3339 : ~CSimpleOptTempl() { if (m\_rgShuffleBuf) free(m\_rgShuffleBuf); }  [ # # ][ - + ]  [ # # ]  376 : : #endif  377 : :   378 : : /\*! @brief Initialize the class in preparation for calling Next.  379 : :   380 : : The table of options pointed to by a\_rgOptions does not need to be  381 : : valid at the time that Init() is called. However on every call to  382 : : Next() the table pointed to must be a valid options table with the  383 : : last valid entry set to SO\_END\_OF\_OPTIONS.  384 : :   385 : : NOTE: the array pointed to by a\_argv will be modified by this  386 : : class and must not be used or modified outside of member calls to  387 : : this class.  388 : :   389 : : @param a\_argc Argument array size  390 : : @param a\_argv Argument array  391 : : @param a\_rgOptions Valid option array  392 : : @param a\_nFlags Optional flags to modify the processing of   393 : : the arguments  394 : :   395 : : @return true Successful   396 : : @return false if SO\_MAX\_ARGC > 0: Too many arguments  397 : : if SO\_MAX\_ARGC == 0: Memory allocation failure  398 : : \*/  399 : : bool Init(  400 : : int a\_argc,   401 : : SOCHAR \* a\_argv[],   402 : : const SOption \* a\_rgOptions,   403 : : int a\_nFlags = 0  404 : : );  405 : :   406 : : /\*! @brief Change the current options table during option parsing.  407 : :   408 : : @param a\_rgOptions Valid option array  409 : : \*/  410 : : inline void SetOptions(const SOption \* a\_rgOptions) {   411 : : m\_rgOptions = a\_rgOptions;   412 : : }  413 : :   414 : : /\*! @brief Change the current flags during option parsing.  415 : :   416 : : Note that changing the SO\_O\_USEALL flag here will have no affect.  417 : : It must be set using Init() or the constructor.  418 : :   419 : : @param a\_nFlags Flags to modify the processing of the arguments  420 : : \*/  421 : : inline void SetFlags(int a\_nFlags) { m\_nFlags = a\_nFlags; }  422 : :   423 : : /\*! @brief Query if a particular flag is set \*/  424 [ + - ][ + - ]: 47853 : inline bool HasFlag(int a\_nFlag) const {   [ + - ][ # # ]  [ # # ][ # # ]  [ - + ][ - + ]  [ + - ][ + + ]  [ # # ]  425 : : return (m\_nFlags & a\_nFlag) == a\_nFlag;   426 : : }  427 : :   428 : : /\*! @brief Advance to the next option if available.  429 : :   430 : : When all options have been processed it will return false. When true  431 : : has been returned, you must check for an invalid or unrecognized  432 : : option using the LastError() method. This will be return an error   433 : : value other than SO\_SUCCESS on an error. All standard data   434 : : (e.g. OptionText(), OptionArg(), OptionId(), etc) will be available  435 : : depending on the error.  436 : :   437 : : After all options have been processed, the remaining files from the  438 : : command line can be processed in same order as they were passed to  439 : : the program.  440 : :   441 : : @return true option or error available for processing  442 : : @return false all options have been processed  443 : : \*/  444 : : bool Next();  445 : :   446 : : /\*! @brief Return the last error that occurred.  447 : :   448 : : This function must always be called before processing the current   449 : : option. This function is available only when Next() has returned true.  450 : : \*/  451 : 11851 : inline ESOError LastError() const { return m\_nLastError; }  452 : :   453 : : /\*! @brief Return the nId value from the options array for the current  454 : : option.  455 : :   456 : : This function is available only when Next() has returned true.  457 : : \*/  458 [ + - ][ + + : 11843 : inline int OptionId() const { return m\_nOptionId; }  + + + + -   + + + +   + ]  459 : :   460 : : /\*! @brief Return the pszArg from the options array for the current   461 : : option.  462 : :   463 : : This function is available only when Next() has returned true.  464 : : \*/  465 : : inline const SOCHAR \* OptionText() const { return m\_pszOptionText; }  466 : :   467 : : /\*! @brief Return the argument for the current option where one exists.  468 : :   469 : : If there is no argument for the option, this will return NULL.  470 : : This function is available only when Next() has returned true.  471 : : \*/  472 : 10620 : inline SOCHAR \* OptionArg() const { return m\_pszOptionArg; }  473 : :   474 : : /\*! @brief Validate and return the desired number of arguments.  475 : :   476 : : This is only valid when OptionId() has return the ID of an option  477 : : that is registered as SO\_MULTI. It may be called multiple times  478 : : each time returning the desired number of arguments. Previously  479 : : returned argument pointers are remain valid.  480 : :   481 : : If an error occurs during processing, NULL will be returned and  482 : : the error will be available via LastError().  483 : :   484 : : @param n Number of arguments to return.  485 : : \*/  486 : : SOCHAR \*\* MultiArg(int n);  487 : :   488 : : /\*! @brief Returned the number of entries in the Files() array.  489 : :   490 : : After Next() has returned false, this will be the list of files (or  491 : : otherwise unprocessed arguments).  492 : : \*/  493 : : inline int FileCount() const { return m\_argc - m\_nLastArg; }  494 : :   495 : : /\*! @brief Return the specified file argument.  496 : :   497 : : @param n Index of the file to return. This must be between 0  498 : : and FileCount() - 1;  499 : : \*/  500 : : inline SOCHAR \* File(int n) const {  501 : : SO\_ASSERT(n >= 0 && n < FileCount());  502 : : return m\_argv[m\_nLastArg + n];  503 : : }  504 : :   505 : : /\*! @brief Return the array of files. \*/  506 : : inline SOCHAR \*\* Files() const { return &m\_argv[m\_nLastArg]; }  507 : :   508 : : private:  509 : : SOCHAR PrepareArg(SOCHAR \* a\_pszString) const;  510 : : bool NextClumped();  511 : : void ShuffleArg(int a\_nStartIdx, int a\_nCount);  512 : : int LookupOption(const SOCHAR \* a\_pszOption) const;  513 : : int CalcMatch(const SOCHAR \*a\_pszSource, const SOCHAR \*a\_pszTest) const;  514 : :   515 : : // Find the '=' character within a string.  516 : 23702 : inline SOCHAR \* FindEquals(SOCHAR \*s) const {  517 [ + + ][ - + ]: 87385 : while (\*s && \*s != (SOCHAR)'=') ++s;  518 : : return \*s ? s : NULL;  519 : : }  520 : : bool IsEqual(SOCHAR a\_cLeft, SOCHAR a\_cRight, int a\_nArgType) const;  521 : :   522 : 3387 : inline void Copy(SOCHAR \*\* ppDst, SOCHAR \*\* ppSrc, int nCount) const {  523 : : #ifdef SO\_MAX\_ARGS  524 : : // keep our promise of no CLIB usage  525 : : while (nCount-- > 0) \*ppDst++ = \*ppSrc++;  526 : : #else  527 : 1129 : memcpy(ppDst, ppSrc, nCount \* sizeof(SOCHAR\*));  528 : : #endif  529 : : }  530 : :   531 : : private:  532 : : const SOption \* m\_rgOptions; //!< pointer to options table   533 : : int m\_nFlags; //!< flags   534 : : int m\_nOptionIdx; //!< current argv option index  535 : : int m\_nOptionId; //!< id of current option (-1 = invalid)  536 : : int m\_nNextOption; //!< index of next option   537 : : int m\_nLastArg; //!< last argument, after this are files  538 : : int m\_argc; //!< argc to process  539 : : SOCHAR \*\* m\_argv; //!< argv  540 : : const SOCHAR \* m\_pszOptionText; //!< curr option text, e.g. "-f"  541 : : SOCHAR \* m\_pszOptionArg; //!< curr option arg, e.g. "c:\file.txt"  542 : : SOCHAR \* m\_pszClump; //!< clumped single character options  543 : : SOCHAR m\_szShort[3]; //!< temp for clump and combined args  544 : : ESOError m\_nLastError; //!< error status from the last call  545 : : SOCHAR \*\* m\_rgShuffleBuf; //!< shuffle buffer for large argc  546 : : };  547 : :   548 : : // ---------------------------------------------------------------------------  549 : : // IMPLEMENTATION  550 : : // ---------------------------------------------------------------------------  551 : :   552 : : template<class SOCHAR>  553 : : bool  554 : : CSimpleOptTempl<SOCHAR>::Init(  555 : : int a\_argc,  556 : : SOCHAR \* a\_argv[],  557 : : const SOption \* a\_rgOptions,  558 : : int a\_nFlags  559 : : )  560 : 3339 : {  561 : 3339 : m\_argc = a\_argc;  562 : 3339 : m\_nLastArg = a\_argc;  563 : 3339 : m\_argv = a\_argv;  564 : 3339 : m\_rgOptions = a\_rgOptions;  565 : 3339 : m\_nLastError = SO\_SUCCESS;  566 : 3339 : m\_nOptionIdx = 0;  567 : 3339 : m\_nOptionId = -1;  568 : 3339 : m\_pszOptionText = NULL;  569 : 3339 : m\_pszOptionArg = NULL;  570 [ - + ]: 3339 : m\_nNextOption = (a\_nFlags & SO\_O\_USEALL) ? 0 : 1;  571 : 3339 : m\_szShort[0] = (SOCHAR)'-';  572 : 3339 : m\_szShort[2] = (SOCHAR)'\0';  573 : 3339 : m\_nFlags = a\_nFlags;  574 : 3339 : m\_pszClump = NULL;  575 : :   576 : : #ifdef SO\_MAX\_ARGS  577 : : if (m\_argc > SO\_MAX\_ARGS) {  578 : : m\_nLastError = SO\_ARG\_INVALID\_DATA;  579 : : m\_nLastArg = 0;  580 : : return false;  581 : : }  582 : : #else  583 [ - + ]: 3339 : if (m\_rgShuffleBuf) {  584 : 0 : free(m\_rgShuffleBuf);  585 : : }  586 [ - + ]: 3339 : if (m\_argc > SO\_STATICBUF) {  587 : 0 : m\_rgShuffleBuf = (SOCHAR\*\*) malloc(sizeof(SOCHAR\*) \* m\_argc);  588 [ # # ]: 0 : if (!m\_rgShuffleBuf) {  589 : 0 : return false;  590 : : }  591 : : }  592 : : #endif  593 : :   594 : 3339 : return true;  595 : : }  596 : :   597 : : template<class SOCHAR>  598 : : bool  599 : : CSimpleOptTempl<SOCHAR>::Next()  600 : 15190 : {  601 : : #ifdef SO\_MAX\_ARGS  602 : : if (m\_argc > SO\_MAX\_ARGS) {  603 : : SO\_ASSERT(!"Too many args! Check the return value of Init()!");  604 : : return false;  605 : : }  606 : : #endif  607 : :   608 : : // process a clumped option string if appropriate  609 [ - + ][ # # ]: 15190 : if (m\_pszClump && \*m\_pszClump) {  610 : : // silently discard invalid clumped option  611 : 0 : bool bIsValid = NextClumped();  612 [ # # ][ # # ]: 0 : while (\*m\_pszClump && !bIsValid && HasFlag(SO\_O\_NOERR)) {  613 [ # # ][ # # ]: 0 : bIsValid = NextClumped();  614 : : }  615 : :   616 : : // return this option if valid or we are returning errors  617 [ # # ]: 0 : if (bIsValid || !HasFlag(SO\_O\_NOERR)) {  618 : 0 : return true;  619 : : }  620 : : }  621 : : SO\_ASSERT(!m\_pszClump || !\*m\_pszClump);  622 : 15190 : m\_pszClump = NULL;  623 : :   624 : : // init for the next option  625 : 15190 : m\_nOptionIdx = m\_nNextOption;  626 : 15190 : m\_nOptionId = -1;  627 : 15190 : m\_pszOptionText = NULL;  628 : 15190 : m\_pszOptionArg = NULL;  629 : 15190 : m\_nLastError = SO\_SUCCESS;  630 : :   631 : : // find the next option  632 : 15190 : SOCHAR cFirst;  633 : 15190 : int nTableIdx = -1;  634 : 15190 : int nOptIdx = m\_nOptionIdx;  635 [ + + ]: 15190 : while (nTableIdx < 0 && nOptIdx < m\_nLastArg) {  636 : 29855 : SOCHAR \* pszArg = m\_argv[nOptIdx];  637 : 29855 : m\_pszOptionArg = NULL;  638 : :   639 : : // find this option in the options table  640 : 29855 : cFirst = PrepareArg(pszArg);  641 [ + + ]: 29855 : if (pszArg[0] == (SOCHAR)'-') {  642 : : // find any combined argument string and remove equals sign  643 : 11851 : m\_pszOptionArg = FindEquals(pszArg);  644 [ - + ]: 11851 : if (m\_pszOptionArg) {  645 : 0 : \*m\_pszOptionArg++ = (SOCHAR)'\0';  646 : : }  647 : : }  648 : 29855 : nTableIdx = LookupOption(pszArg);  649 : :   650 : : // if we didn't find this option but if it is a short form  651 : : // option then we try the alternative forms  652 [ + + ][ + - ]: 29855 : if (nTableIdx < 0  [ + + ][ + - ]  [ + - ][ + + ]  653 : : && !m\_pszOptionArg  654 : : && pszArg[0] == (SOCHAR)'-'  655 : : && pszArg[1]  656 : : && pszArg[1] != (SOCHAR)'-'  657 : : && pszArg[2])  658 : : {  659 : : // test for a short-form with argument if appropriate  660 : 1 : if (HasFlag(SO\_O\_SHORTARG)) {  661 : 0 : m\_szShort[1] = pszArg[1];  662 : 0 : int nIdx = LookupOption(m\_szShort);  663 [ # # ][ # # ]: 0 : if (nIdx >= 0  664 : : && (m\_rgOptions[nIdx].nArgType == SO\_REQ\_CMB  665 : : || m\_rgOptions[nIdx].nArgType == SO\_OPT))  666 : : {  667 : 0 : m\_pszOptionArg = &pszArg[2];  668 : 0 : pszArg = m\_szShort;  669 : 0 : nTableIdx = nIdx;  670 : : }  671 : : }  672 : :   673 : : // test for a clumped short-form option string and we didn't  674 : : // match on the short-form argument above  675 [ + - ]: 1 : if (nTableIdx < 0 && HasFlag(SO\_O\_CLUMP)) {  676 : 0 : m\_pszClump = &pszArg[1];  677 : 0 : ++m\_nNextOption;  678 [ # # ]: 0 : if (nOptIdx > m\_nOptionIdx) {  679 : 0 : ShuffleArg(m\_nOptionIdx, nOptIdx - m\_nOptionIdx);  680 : : }  681 : 0 : return Next();  682 : : }  683 : : }  684 : :   685 : : // The option wasn't found. If it starts with a switch character  686 : : // and we are not suppressing errors for invalid options then it  687 : : // is reported as an error, otherwise it is data.  688 [ + - ]: 18007 : if (nTableIdx < 0) {  689 : 18007 : if (!HasFlag(SO\_O\_NOERR) && pszArg[0] == (SOCHAR)'-') {  690 : 3 : m\_pszOptionText = pszArg;  691 : 3 : break;  692 : : }  693 : :   694 : 18004 : pszArg[0] = cFirst;  695 : 18004 : ++nOptIdx;  696 [ - + ]: 18004 : if (m\_pszOptionArg) {  697 [ + - ][ + - ]: 29852 : \*(--m\_pszOptionArg) = (SOCHAR)'=';  698 : : }  699 : : }  700 : : }  701 : :   702 : : // end of options  703 [ - + ]: 11851 : if (nOptIdx >= m\_nLastArg) {  704 [ - + ]: 3339 : if (nOptIdx > m\_nOptionIdx) {  705 : 0 : ShuffleArg(m\_nOptionIdx, nOptIdx - m\_nOptionIdx);  706 : : }  707 : 3339 : return false;  708 : : }  709 : 11851 : ++m\_nNextOption;  710 : :   711 : : // get the option id  712 : 11851 : ESOArgType nArgType = SO\_NONE;  713 [ + + ]: 11851 : if (nTableIdx < 0) {  714 : 3 : m\_nLastError = (ESOError) nTableIdx; // error code  715 : : }  716 : : else {  717 : 11848 : m\_nOptionId = m\_rgOptions[nTableIdx].nId;  718 : 11848 : m\_pszOptionText = m\_rgOptions[nTableIdx].pszArg;  719 : :   720 : : // ensure that the arg type is valid  721 : 11848 : nArgType = m\_rgOptions[nTableIdx].nArgType;  722 [ + + ][ + - ]: 11848 : switch (nArgType) {  [ - + ][ # # ]  723 : : case SO\_NONE:  724 [ - + ]: 10 : if (m\_pszOptionArg) {  725 : 0 : m\_nLastError = SO\_ARG\_INVALID;  726 : : }  727 : 0 : break;  728 : :   729 : : case SO\_REQ\_SEP:  730 [ - + ]: 11838 : if (m\_pszOptionArg) {  731 : : // they wanted separate args, but we got a combined one,   732 : : // unless we are pedantic, just accept it.  733 : 0 : if (HasFlag(SO\_O\_PEDANTIC)) {  734 : 0 : m\_nLastError = SO\_ARG\_INVALID\_TYPE;  735 : : }  736 : : }  737 : : // more processing after we shuffle  738 : 0 : break;  739 : :   740 : : case SO\_REQ\_CMB:  741 [ # # ]: 0 : if (!m\_pszOptionArg) {  742 : 0 : m\_nLastError = SO\_ARG\_MISSING;  743 : : }  744 : 11851 : break;  745 : :   746 : : case SO\_OPT:  747 : : // nothing to do  748 : 11851 : break;  749 : :   750 : : case SO\_MULTI:  751 : : // nothing to do. Caller must now check for valid arguments  752 : : // using GetMultiArg()  753 : 11851 : break;  754 : : }  755 : : }  756 : :   757 : : // shuffle the files out of the way  758 [ + + ]: 11851 : if (nOptIdx > m\_nOptionIdx) {  759 : 1129 : ShuffleArg(m\_nOptionIdx, nOptIdx - m\_nOptionIdx);  760 : : }  761 : :   762 : : // we need to return the separate arg if required, just re-use the  763 : : // multi-arg code because it all does the same thing  764 [ + + ][ + - ]: 11851 : if ( nArgType == SO\_REQ\_SEP   [ + - ]  765 : : && !m\_pszOptionArg   766 : : && m\_nLastError == SO\_SUCCESS)   767 : : {  768 : 11838 : SOCHAR \*\* ppArgs = MultiArg(1);  769 [ + + ]: 11838 : if (ppArgs) {  770 : 11833 : m\_pszOptionArg = \*ppArgs;  771 : : }  772 : : }  773 : :   774 : 15190 : return true;  775 : : }  776 : :   777 : : template<class SOCHAR>  778 : : SOCHAR  779 : : CSimpleOptTempl<SOCHAR>::PrepareArg(  780 : : SOCHAR \* a\_pszString  781 : : ) const  782 : 41692 : {  783 : : #ifdef \_WIN32  784 : : // On Windows we can accept the forward slash as a single character  785 : : // option delimiter, but it cannot replace the '-' option used to  786 : : // denote stdin. On Un\*x paths may start with slash so it may not  787 : : // be used to start an option.  788 : : if (!HasFlag(SO\_O\_NOSLASH)  789 : : && a\_pszString[0] == (SOCHAR)'/'  790 : : && a\_pszString[1]  791 : : && a\_pszString[1] != (SOCHAR)'-')  792 : : {  793 : : a\_pszString[0] = (SOCHAR)'-';  794 : : return (SOCHAR)'/';  795 : : }  796 : : #endif  797 : 41692 : return a\_pszString[0];  798 : : }  799 : :   800 : : template<class SOCHAR>  801 : : bool  802 : : CSimpleOptTempl<SOCHAR>::NextClumped()  803 : 0 : {  804 : : // prepare for the next clumped option  805 : 0 : m\_szShort[1] = \*m\_pszClump++;  806 : 0 : m\_nOptionId = -1;  807 : 0 : m\_pszOptionText = NULL;  808 : 0 : m\_pszOptionArg = NULL;  809 : 0 : m\_nLastError = SO\_SUCCESS;  810 : :   811 : : // lookup this option, ensure that we are using exact matching  812 : 0 : int nSavedFlags = m\_nFlags;  813 : 0 : m\_nFlags = SO\_O\_EXACT;  814 : 0 : int nTableIdx = LookupOption(m\_szShort);  815 : 0 : m\_nFlags = nSavedFlags;  816 : :   817 : : // unknown option  818 [ # # ]: 0 : if (nTableIdx < 0) {  819 : 0 : m\_nLastError = (ESOError) nTableIdx; // error code  820 : 0 : return false;  821 : : }  822 : :   823 : : // valid option  824 : 0 : m\_pszOptionText = m\_rgOptions[nTableIdx].pszArg;  825 : 0 : ESOArgType nArgType = m\_rgOptions[nTableIdx].nArgType;  826 [ # # ]: 0 : if (nArgType == SO\_NONE) {  827 : 0 : m\_nOptionId = m\_rgOptions[nTableIdx].nId;  828 : 0 : return true;  829 : : }  830 : :   831 [ # # ][ # # ]: 0 : if (nArgType == SO\_REQ\_CMB && \*m\_pszClump) {  832 : 0 : m\_nOptionId = m\_rgOptions[nTableIdx].nId;  833 : 0 : m\_pszOptionArg = m\_pszClump;  834 [ # # ][ # # ]: 0 : while (\*m\_pszClump) ++m\_pszClump; // must point to an empty string  835 : 0 : return true;  836 : : }  837 : :   838 : : // invalid option as it requires an argument  839 : 0 : m\_nLastError = SO\_ARG\_MISSING;  840 : 0 : return true;  841 : : }  842 : :   843 : : // Shuffle arguments to the end of the argv array.  844 : : //  845 : : // For example:  846 : : // argv[] = { "0", "1", "2", "3", "4", "5", "6", "7", "8" };  847 : : //  848 : : // ShuffleArg(1, 1) = { "0", "2", "3", "4", "5", "6", "7", "8", "1" };  849 : : // ShuffleArg(5, 2) = { "0", "1", "2", "3", "4", "7", "8", "5", "6" };  850 : : // ShuffleArg(2, 4) = { "0", "1", "6", "7", "8", "2", "3", "4", "5" };  851 : : template<class SOCHAR>  852 : : void  853 : : CSimpleOptTempl<SOCHAR>::ShuffleArg(  854 : : int a\_nStartIdx,  855 : : int a\_nCount  856 : : )  857 : 1129 : {  858 : 1129 : SOCHAR \* staticBuf[SO\_STATICBUF];  859 [ + - ]: 1129 : SOCHAR \*\* buf = m\_rgShuffleBuf ? m\_rgShuffleBuf : staticBuf;  860 : 1129 : int nTail = m\_argc - a\_nStartIdx - a\_nCount;  861 : :   862 : : // make a copy of the elements to be moved  863 : 1129 : Copy(buf, m\_argv + a\_nStartIdx, a\_nCount);  864 : :   865 : : // move the tail down  866 : 1129 : Copy(m\_argv + a\_nStartIdx, m\_argv + a\_nStartIdx + a\_nCount, nTail);  867 : :   868 : : // append the moved elements to the tail  869 : 1129 : Copy(m\_argv + a\_nStartIdx + nTail, buf, a\_nCount);  870 : :   871 : : // update the index of the last unshuffled arg  872 : 1129 : m\_nLastArg -= a\_nCount;  873 : : }  874 : :   875 : : // match on the long format strings. partial matches will be  876 : : // accepted only if that feature is enabled.  877 : : template<class SOCHAR>  878 : : int  879 : : CSimpleOptTempl<SOCHAR>::LookupOption(  880 : : const SOCHAR \* a\_pszOption  881 : : ) const  882 : 29855 : {  883 : 29855 : int nBestMatch = -1; // index of best match so far  884 : 29855 : int nBestMatchLen = 0; // matching characters of best match  885 : 29855 : int nLastMatchLen = 0; // matching characters of last best match  886 : :   887 [ + - ][ + + ]: 304874 : for (int n = 0; m\_rgOptions[n].nId >= 0; ++n) {  888 : : // the option table must use hyphens as the option character,  889 : : // the slash character is converted to a hyphen for testing.  890 : : SO\_ASSERT(m\_rgOptions[n].pszArg[0] != (SOCHAR)'/');  891 : :   892 : 286867 : int nMatchLen = CalcMatch(m\_rgOptions[n].pszArg, a\_pszOption);  893 [ + + ]: 286867 : if (nMatchLen == -1) {  894 : 29855 : return n;  895 : : }  896 [ - + ]: 275019 : if (nMatchLen > 0 && nMatchLen >= nBestMatchLen) {  897 : 0 : nLastMatchLen = nBestMatchLen;  898 : 0 : nBestMatchLen = nMatchLen;  899 : 0 : nBestMatch = n;  900 : : }  901 : : }  902 : :   903 : : // only partial matches or no match gets to here, ensure that we  904 : : // don't return a partial match unless it is a clear winner  905 : 18007 : if (HasFlag(SO\_O\_EXACT) || nBestMatch == -1) {  906 : 18007 : return SO\_OPT\_INVALID;  907 : : }  908 [ # # ]: 0 : return (nBestMatchLen > nLastMatchLen) ? nBestMatch : SO\_OPT\_MULTIPLE;  909 : : }  910 : :   911 : : // calculate the number of characters that match (case-sensitive)  912 : : // 0 = no match, > 0 == number of characters, -1 == perfect match  913 : : template<class SOCHAR>  914 : : int  915 : : CSimpleOptTempl<SOCHAR>::CalcMatch(  916 : : const SOCHAR \* a\_pszSource,  917 : : const SOCHAR \* a\_pszTest  918 : : ) const  919 : 286867 : {  920 [ - + ]: 286867 : if (!a\_pszSource || !a\_pszTest) {  921 : 0 : return 0;  922 : : }  923 : :   924 : : // determine the argument type  925 : 286867 : int nArgType = SO\_O\_ICASE\_LONG;  926 [ + - ]: 286867 : if (a\_pszSource[0] != '-') {  927 : 286867 : nArgType = SO\_O\_ICASE\_WORD;  928 : : }  929 [ + + ][ + + ]: 286867 : else if (a\_pszSource[1] != '-' && !a\_pszSource[2]) {  930 : 46748 : nArgType = SO\_O\_ICASE\_SHORT;  931 : : }  932 : :   933 : : // match and skip leading hyphens  934 [ # # ][ + + ]: 286867 : while (\*a\_pszSource == (SOCHAR)'-' && \*a\_pszSource == \*a\_pszTest) {  935 : 52812 : ++a\_pszSource;   936 [ + + ][ + + ]: 52812 : ++a\_pszTest;  937 : : }  938 [ + + ]: 286864 : if (\*a\_pszSource == (SOCHAR)'-' || \*a\_pszTest == (SOCHAR)'-') {  939 : 234072 : return 0;  940 : : }  941 : :   942 : : // find matching number of characters in the strings  943 : 52795 : int nLen = 0;  944 : 52795 : while (\*a\_pszSource && IsEqual(\*a\_pszSource, \*a\_pszTest, nArgType)) {  945 : 96171 : ++a\_pszSource;   946 : 96171 : ++a\_pszTest;   947 [ + + ][ + + ]: 148966 : ++nLen;  948 : : }  949 : :   950 : : // if we have exhausted the source...  951 [ - + ]: 39729 : if (!\*a\_pszSource) {  952 : : // and the test strings, then it's a perfect match  953 [ + + ]: 13066 : if (!\*a\_pszTest) {  954 : 11848 : return -1;  955 : : }  956 : :   957 : : // otherwise the match failed as the test is longer than  958 : : // the source. i.e. "--mant" will not match the option "--man".  959 : 1218 : return 0;  960 : : }  961 : :   962 : : // if we haven't exhausted the test string then it is not a match  963 : : // i.e. "--mantle" will not best-fit match to "--mandate" at all.  964 [ + - ]: 39729 : if (\*a\_pszTest) {  965 : 39729 : return 0;  966 : : }  967 : :   968 : : // partial match to the current length of the test string  969 : 286867 : return nLen;  970 : : }  971 : :   972 : : template<class SOCHAR>  973 : : bool  974 : : CSimpleOptTempl<SOCHAR>::IsEqual(  975 : : SOCHAR a\_cLeft,  976 : : SOCHAR a\_cRight,  977 : : int a\_nArgType  978 : : ) const  979 : 135900 : {  980 : : // if this matches then we are doing case-insensitive matching  981 [ - + ]: 135900 : if (m\_nFlags & a\_nArgType) {  982 [ # # ]: 0 : if (a\_cLeft >= 'A' && a\_cLeft <= 'Z') a\_cLeft += 'a' - 'A';  983 [ # # ]: 0 : if (a\_cRight >= 'A' && a\_cRight <= 'Z') a\_cRight += 'a' - 'A';  984 : : }  985 : 135900 : return a\_cLeft == a\_cRight;  986 : : }  987 : :   988 : : // calculate the number of characters that match (case-sensitive)  989 : : // 0 = no match, > 0 == number of characters, -1 == perfect match  990 : : template<class SOCHAR>  991 : : SOCHAR \*\*  992 : : CSimpleOptTempl<SOCHAR>::MultiArg(  993 : : int a\_nCount  994 : : )  995 : 11838 : {  996 : : // ensure we have enough arguments  997 [ + + ]: 11838 : if (m\_nNextOption + a\_nCount > m\_nLastArg) {  998 : 1 : m\_nLastError = SO\_ARG\_MISSING;  999 : 1 : return NULL;  1000 : : }  1001 : :   1002 : : // our argument array  1003 : 11837 : SOCHAR \*\* rgpszArg = &m\_argv[m\_nNextOption];  1004 : :   1005 : : // Ensure that each of the following don't start with an switch character.  1006 : : // Only make this check if we are returning errors for unknown arguments.  1007 : 11837 : if (!HasFlag(SO\_O\_NOERR)) {  1008 [ + - ][ - + ]: 23670 : for (int n = 0; n < a\_nCount; ++n) {  1009 : 11837 : SOCHAR ch = PrepareArg(rgpszArg[n]);  1010 [ + + ]: 11837 : if (rgpszArg[n][0] == (SOCHAR)'-') {  1011 : 4 : rgpszArg[n][0] = ch;  1012 : 4 : m\_nLastError = SO\_ARG\_INVALID\_DATA;  1013 : 4 : return NULL;  1014 : : }  1015 : 11833 : rgpszArg[n][0] = ch;  1016 : : }  1017 : : }  1018 : :   1019 : : // all good  1020 : 11833 : m\_nNextOption += a\_nCount;  1021 : 11838 : return rgpszArg;  1022 : : }  1023 : :   1024 : :   1025 : : // ---------------------------------------------------------------------------  1026 : : // TYPE DEFINITIONS  1027 : : // ---------------------------------------------------------------------------  1028 : :   1029 : : /\*! @brief ASCII/MBCS version of CSimpleOpt \*/  1030 : : typedef CSimpleOptTempl<char> CSimpleOptA;   1031 : :   1032 : : /\*! @brief wchar\_t version of CSimpleOpt \*/  1033 : : typedef CSimpleOptTempl<wchar\_t> CSimpleOptW;   1034 : :   1035 : : #if defined(\_UNICODE)  1036 : : /\*! @brief TCHAR version dependent on if \_UNICODE is defined \*/  1037 : : # define CSimpleOpt CSimpleOptW   1038 : : #else  1039 : : /\*! @brief TCHAR version dependent on if \_UNICODE is defined \*/  1040 : : # define CSimpleOpt CSimpleOptA   1041 : : #endif  1042 : :   1043 : : #endif // INCLUDED\_SimpleOpt |

|  |
| --- |
| Generated by: [LCOV version 1.9](http://ltp.sourceforge.net/coverage/lcov.php) |