Edit Env Reference Manual

0.1.0-rc3 (29 apr 2003)

Generated by Doxygen 1.3

Tue Apr 29 11:20:45 2003

Contents

Chapter 1

editenv – display, create, change or delete environment variable.

Introduction

editenv is a program to handle environment variables for Windows NT (registry) and Windows 95 (`autoexec.bat`) types of operating systems. See the manual page for more information on how to use the program. To learn more about the program's internal setup, see the editenv implementation section.

editenv License

editenv is written by Martin J. Moene <moene@biophys.LeidenUniv.nl>

Chapter 2

Edit Env Module Index

2.1 Edit Env Modules

Here is a list of all modules:

Chapter 3

Edit Env Page Index

3.1 Edit Env Related Pages

Here is a list of all related documentation pages:

Chapter 4

Edit Env Module Documentation

4.1 editenv – manual page.

```
NAME
       editenv - display, create, change or delete environment variable.


SYNOPSIS
       editenv [-hHvV] [-cdepu] [var[=value] ...]

DESCRIPTION
      editenv lists, creates, changes or deletes environment variables in the
      registry on Windows NT type operating systems (Windows NT/2000/XP) and
      in autoexec.bat on Windows 95 type operating systems (Windows 95/98/Me).

   Options
      editenv can be executed with the following options:

      -a   print author information
      -h   print overview of options
      -H   print program description
      -c   edit autoexec.bat in current dir. (Win95)
      -d   delete specified variable
      -e   use current environment
      -u   env. variable in user hive (WinNT)
      -p   print platform information
      -v+  verbose; print progress information (-v: stdout, -V:stderr)
      --   end option section

   Processing
      If the operating system is Windows NT type, the specified environment
      variable is created or updated in or deleted from the registry.
      If option -u is specified, it is done in the 'current user' part of
      the registry, otherwise it is done in the 'local machine' part.

      If the operating system is Windows 95 type, autoexec.bat is edited,
      or created if none exists. First autoexec.bat is scanned to look for
      ocurrences of the specified variable and if found, the last location
      is remembered. Then a new autoexec.bat is created with the environment
      variable updated or removed. Note: if multiple definitions of the
      variable exist, the last but one becomes active again after removal
      of the last occurrence.

   Error messages
      Most error messages editenv can issue, concern file operations that
      fail.
```

```
   Program exit status
      When a file cannot be properly processed, the program stops and
      issues an error message. The failure to process a file is reflected
      in the programs exit status (see DIAGNOSTICS below).


ENVIRONMENT
       none.


FILES
      editenv uses and creates the following files:

      c:\\autoexec.bat  on Windows 95 type operating system this file is
                        created, or edited to add, remove, or change the
                        specified environment variable(s).

      c:\\autoexec.new  see c:\\autoexec.bat. this file holds the contents
                        of the autoexec.bat being build.

      c:\\autoexec.sav  see c:\\autoexec.bat. this file holds the contents
                        of the last autoexec.bat


DIAGNOSTICS
      editenv can return the following exit values:

      0 success: program execution has been successfully completed,
      1 commandline error: an invalid option is specified,
      2 processing error: a file could not be opened or closed, an error
        occurred while writing to an output file,
      3 interruption: the user interrupted the program,
      4 internal error: an unexpected situation in program behaviour occurred.


SEE ALSO
      (none)


LIMITATIONS
      on Windows 95 type operating systems it is assumed that the file
      autotexec.bat is located in c:\.


BUGS
      (to be determined.)


AUTHOR
      M.J. Moene (moene@biophys.LeidenUniv.nl)

      registry editing part inspired on code by Arno C.J. van Amersfoort

      Huygens Laboratorium
      Niels Bohrweg 2
      2333 CA Leiden
      The Netherlands
```

4.2 editenv – implementation.

4.2.1 Detailed Description

editenv contains a function main() that collects the options from the commandline and loops through the remaining non-option arguments.

The arguments can have two forms:

1. **varname**

   is used to delete an environment variable (option `-d`) or to display the contents of a variable (default behaviour).

2. **varname="a value"**

   is used to assign contents to an environment variable. The quotation marks may be omitted if the value to assign consists of a single word.

If the program is running on a Windows NT type of operating system (Windows NT/2000/XP), function doWinNT() is called to query or edit the registry. On Windows 95 type of operating systems (Windows 95/98/me), function doWin95() is called to query or edit the file `autoexec.bat`. However, if optCurEnv has been set (option `-e`), the current environment is used instead of the registry or `autoexec.bat`.

Information about the Win32 API is obtained from the Microsoft®Win32®Programmer's Reference (Win32.hlp) help file as distributed with Borland C++ version 5.

Namespaces

- namespace std

Typedefs

- typedef void(∗ signal_fptr)()

  Ctrl-C (user interrupt) handler function type.

Functions

- int author (int status, FILE ∗stream)

  print author contact information to stream, return status.

- int usage (int status, FILE ∗stream)

  print intent and usage of applicationto stream, return status.

- int manual (int status, FILE ∗stream)

  print a description of this program to stream, return status.

- int platform (int status, FILE ∗stream)

  print platform information to stream, return status.

- int doWinNT (const string &cmd, const string &name, string &value)

set, change or delete environment variable on Windows NT type (Windows NT/2000/XP).

- int doWin95 (const string &cmd, const string &name, string &value)
  set, change or delete environment variable on Windows 95 type (Windows 95/98/Me).
- int isWinNT ()
  return true if running on Windows NT type OS (Windows NT/2000/XP).
- int isWin95 ()
  return true if running on Windows 95 type OS (Windows 95/98/Me).
- int splitArg (const string &arg, string &cmd, string &name, string &value)
  split arg into (name,command,value).
- void _USERENTRY catcher ()
  Ctrl-C (user interrupt) handler.
- int main (int argc, char *argv[ ])
  handle commandline arguments.
- int readCurEnv (const string &name, string &value)
  read variable from current program's environment.
- int writeCurEnv (const string &name, const string &value)
  write variable to current program's environment.
- int readRegEnv (const string &name, string &value, int cu)
  read environment variable from registry.
- int writeRegEnv (const string &name, const string &value, int cu)
  write environment variable to registry.
- int deleteRegEnv (const string &name, int cu)
  delete environment variable from registry.
- int readIniVar (string name, string &value, string section, string fname)
  read variable from initialization file.
- int writeIniVar (string name, string value, string section, string fname)
  write variable to initialization file. See editIniVar().
- int deleteIniVar (string name, string section, string fname)
  delete variable from initialization file. See editIniVar().
- string replaceExt (const string &name, const string &ext)
  replace filename extension.
- int editIniVar (string name, string value, string section, string fname, int doDelete)
  write variable to or delete variable from initialization file.

Variables

- const char * title

  program banner.

- int optDelete = 0

  option -d: delete environment variable

- int optCurDir = 0

  option -c: autoexec in current dir (Win95)

- int optCurEnv = 0

  option -e: use current environment

- int optKeepReg = 0

  option -k: keep registry editing file (WinNT)

- int optHKCU = 0

  option -u: environment variable in Current User environment (WinNT)

- int optVerbose = 0

  option -v/V: verbose

4.2.2 Function Documentation

4.2.2.1 int deleteIniVar (string *name*, string *section*, string *fname*)

deleteIniVar() deletes variable `name` from the initialization file specified by `fname`. `section` may specify the section to restrict the action to.

However, if a `section` name *is* specified, deleteIniVar() warns that it will be ignored: deleteIniVar() uses editIniVar() that is designed to handle `autoexec.bat`, but is not yet able to handle Windows initialization file `[section]`'s.

See further editIniVar().

4.2.2.2 int deleteRegEnv (const string & *name*, int *cu*)

deleteRegEnv() deletes environment variable `name` from the registry. writeRegEnv() uses Windows API functions RegOpenKeyEx(), RegDeleteValueEx() and RegCloseKey().

Parameters:

name the name of the variable

cu if true, use Current User hive, otherwise use Local Machine hive

Returns:

0, 1 (Ok, error)

4.2.2.3 int doWin95 (const string & *cmd*, const string & *name*, string & *value*) [static]

On Windows 95 type operating systems, file `autoexec.bat` is edited, or created if none exists. First `autoexec.bat` is scanned to look for ocurrences of the specified variable and if found, the last location is remembered. Then a new `autoexec.bat` is created as follows:

- **new** a new environment variable is appended to the file
- **update** the last occurrence of the environment variable in question is updated
- **remove** the last occurrence of the environment variable in question is removed

doWin95() uses functions deleteIniVar(), writeIniVar() and readIniVar() from module `iniio`.cpp to query and edit `autoexec.bat`.

Note 1: if multiple definitions of the variable exist, the last but one becomes active again after removal of the last occurrence.

Note 2: unless option `-c` (current directory) has been specified, it is assumed that `autoexec.bat` resides on drive `C:` .

Note 3: if option `-e` (current environment) has been specified, file `autoexec.bat` is not referenced or changed at all.

Parameters:

cmd one of `[-:=]`, meaning delete, display, assign

name variable name

value value queried or value to assign

Returns:

E_OK, E_OPT, E_PS

See also:

deleteIniVar(), writeIniVar(), readIniVar()

4.2.2.4 int doWinNT (const string & *cmd*, const string & *name*, string & *value*) [static]

On Windows NT type operating systems, the environment information is contained in the registry.

doWinNT() queries and edits the registry via the Windows API functions. To this end it uses functions deleteRegEnv(), writeRegEnv() and readRegEnv() from module `envio`.cpp.

Note that it is also possible to use program `regedit` to edit the registry. For an example of how this can be done, see Edit Registry via Program Regedit.

Parameters:

cmd one of `[-:=]`, meaning delete, display, assign

name variable name

value value queried or value to assign

Returns:

E_OK, E_OPT, E_PS

See also:

deleteRegEnv(), writeRegEnv(), readRegEnv()

4.2.2.5 int editIniVar (string *name*, string *value*, string *section*, string *fname*, int *doDelete*)

editIniVar() looks for the variable specified by `name` in a Windows initializaton (.ini) file, or in the file `autoexec.bat` as specified by `fname` and `section`. Then it

- creates the variable with the `value` specified,
- replaces the variable's contents with the `value` specified, or
- deletes the variable from the file id `doDelete` is true. However, see the note below.

First editIniVar() opens the file specified by `fname`, or create one if it does not yet exist.

Then editIniVar() locates the line(s) with variable `name`.

The lines of interest are searched for by looking at the pattern

```
{whitespace}* [sSeEtT] {whitespace}+ [a-zA-Z0-9]+
```

to read variable names and compare them to the one looked for.

editIniVar() remembers the linenumber of the last occurrence of the variable `name`.

Now the file `fname`.*new* is created to copy the contents of the original file to, skipping the last line with variable `name` if it must be deleted, or replacing that line when a new `value` must be assigned to the variable. If it is a new variable, a new line is simply appended to the new file.

Note that if the edited file contains a script with labels and goto's, a line appended to the end of the file may never be reached.

Finally editEnv() makes the new file available as `fname` and retains a copy of the original file in `fname`.*sav*.

Note:

editIniVar() currently cannot select Windows initialization file `[section]`'s. The `section` specification is ignored.

editIniVar() is used by functions writeIniVar() and deleteIniVar().

Parameters:

name variable name

value the variable value passed back

section the name of a section [name] (currently ignored)

fname name of the initialization file (currently implemented for autoexec.bat)

doDelete if true, delete the variable specified by `name`

Returns:

0, 1 (variable edited, error)

4.2.2.6 int main (int *argc*, char * *argv*[ ])

main() processes the commandline arguments as follows.

If no arguments are specified on the commandline, main() calls usage() to print a short help screen, otherwise main() collects the options from the commandline.

If after collecting the options, there are no arguments left on the commandline specified, main() also calls usage() to print a short description of the program's usage.

Now main() starts handling the commandline arguments (`var`[=value] ...).

For each argument main() tries to split it into a (name, command, value) triple, using function splitArg(). If optDelete has been set (option `-d`), command is set to `-` .

On Windows NT types of operating systems, main() delegates the work to doWinNT() and passes it the triple, On Windows NT types of operating systems, main() delegates the work to doWin95() and passes it the triple.

If main() is deleting or setting an environment variable and doWinNT() or doWin95() returns with an error main() exits, issuing an error message and reminding the user: *"do you have sufficient administrative rights?"*.

If all went well, main() returns E_OK.

Parameters:

argc argument count

argv argument vector

Returns:

see E_OK

Examples:

main-simple.cpp.

4.2.2.7 int readCurEnv (const string & *name*, string & *value*)

readCurEnv() reads the contents of variable `name` from the current program's environment using Windows API function GetEnvironmentVariable() and passes it back in `value`.

If the variable does not exist, readCurEnv() returns 1, and `value` is unchanged. On success readCurEnv() returns 0.

Parameters:

name the name of the variable

value the contents of the variable passed back

Returns:

0, 1 (Ok, no variable `name`)

4.2.2.8 int readIniVar (string *name*, string & *value*, string *section*, string *fname*)

readIniVar() reads the contents of a variable from a Windows initializaton (.ini) file, or from the file `autoexec.bat`. However, see the note below.

If a `section` name is specified, readIniVar() warns that it will be ignored: readIniVar() is designed to handle `autoexec.bat`, but it is not yet able to handle Windows initialization file `[section]`'s.

readIniVar() opens the file specfied by `fname`. It returns 1 if this fails.

Then readIniVar() locates the line(s) with variable `name` and obtains the variable's value from the last occurrence.

The lines of interest are searched for by looking at the pattern

```
{whitespace}* [sSeEtT] {whitespace}+ [a-zA-Z0-9]+
```

to read variable names and compare them to the one looked for.

If the variable is found, readIniVar() reads the `value` with pattern

```
^[^=]=[\\"]?([^\\"\\n]+)
```

When all lines are read, the file is closed. If this fails, readIniVar() returns 1.

Finally readIniVar() returns 0 if the requested variable has been found, or 1 if it could not be read.

Note:

readIniVar() currently cannot select Windows initialization file `[section]`'s. The `section` specification is ignored and a warning message is printed to `stderr`.

Parameters:

name variable name

value the variable value passed back

section the name of a section [name] (currently ignored)

fname name of the initialization file (currently implemented for autoexec.bat)

Returns:

0, 1 (variable value read, error or variable not found)

4.2.2.9 int readRegEnv (const string & *name*, string & *value*, int *cu*)

readRegEnv() reads the contents of environment variable `name` from the registry and passes it back in `value`. readRegEnv() uses Windows API functions RegOpenKeyEx(), RegQueryValueEx() and RegCloseKey().

Parameters:

name the name of the variable

value the contents of the variable passed back

cu if true, use Current User hive, otherwise use Local Machine hive

Returns:

0, 1 (Ok, error)

4.2.2.10 int splitArg (const string & *arg*, string & *cmd*, string & *name*, string & *value*) `[static]`

splitArg() takes the argument in the form `varname`[=value] and decomposes it into a variable name, a command and a value.

In all situations the variable name is passed via the `name` parameter. If only the varname is given, parameter `cmd` is set to `:` indicating *display*. If an assignment is given, parameter `cmd` is set to `=` to indicate *assign* and the value is passed to parameter `value`.

Parameters:

arg the input argument `var`[=value]

cmd command specifying assign (=) or display (:)

name variable name concerned

value value to assign

Returns:
0 (Ok)

4.2.2.11 int writeCurEnv (const string & *name*, const string & *value*)

writeCurEnv() writes `value` as the contents of variable `name` in the current program's environment using Windows API function SetEnvironmentVariable().

Parameters:

name the name of the variable

value the contents of the variable passed back

Returns:
0, 1 (Ok, error)

4.2.2.12 int writeIniVar (string *name*, string *value*, string *section*, string *fname*)

writeIniVar() writes variable `name` with contents `value` to the initialization file specified by `fname`. `section` may specify the section to put the variable in.

However, if a `section` name *is* specified, writeIniVar() warns that it will be ignored: writeIniVar() uses editIniVar() that is designed to handle `autoexec.bat`, but is not yet able to handle Windows initialization file `[section]`'s.

See further editIniVar().

4.2.2.13 int writeRegEnv (const string & *name*, const string & *value*, int *cu*)

writeRegEnv() writes `value` as the new contents of environment variable `name` in the registry. writeRegEnv() uses Windows API functions RegOpenKeyEx(), RegSetValueEx() and RegCloseKey().

Parameters:

name the name of the variable

value the new contents of the variable

cu if true, use Current User hive, otherwise use Local Machine hive

Returns:
0, 1 (Ok, error)

4.2.3 Variable Documentation

4.2.3.1 const char* title

Initial value:

```
"editenv  " VERSION_STRING "  edit environment variables.\n"
"Copyright (C) 2003, Leiden University. All rights reserved.\n"
"editenv is free software, released under the terms of the GNU GPL."
```

4.3 getopt – commandline option handling

4.3.1 Detailed Description

Module Xgetopt is written by Hans Dietrich <hdietrich2@hotmail.com>

Module Xgetopt is released into the public domain. You are free to use it in any way you like.

This software is provided ”as is” with no expressed or implied warranty. I accept no liability for any damage or loss of business that this software may cause.

Functions

- int getopt (int argc, char ∗argv[ ], char ∗optstring)

  parse command line options

Variables

- char ∗ optarg

  optarg points to the current option’s argument.

- int optind = 0

  argv index of next argument to process.

- int opterr

  opterr is not used by this version of getopt().

4.3.2 Function Documentation

4.3.2.1 int getopt (int *argc*, char ∗ *argv*[ ], char ∗ *optstring*)

The getopt() function parses the command line arguments. Its arguments `argc` and `argv` are the argument count and array as passed into the application on program invocation. In the case of Visual C++ programs, `argc` and `argv` are available via the variables `__argc` and `__argv` (double underscores), respectively. getopt() returns the next option letter in `argv` that matches a letter in `optstring`.

`optstring` is a string of recognized option letters; if a letter is followed by a colon, the option is expected to have an argument that may or may not be separated from it by white space. `optarg` is set to point to the start of the option argument on return from getopt().

Option letters may be combined, e.g., `"-ab"` is equivalent to `"-a -b"`. Option letters are case sensitive.

getopt() places in the external variable `optind` the `argv` index of the next argument to be processed. `optind` is initialized to 0 before the first call to getopt().

When all options have been processed (i.e., up to the first non-option argument), getopt() returns `EOF`, `optarg` will point to the argument, and `optind` will be set to the `argv` index of the argument. If there are no non-option arguments, `optarg` will be set to `NULL`.

The special option `"--"` may be used to delimit the end of the options; `EOF` will be returned, and `"--"` (and everything after it) will be skipped.

Returns:

For option letters contained in the string `optstring`, getopt() will return the option letter. getopt() returns a question mark (?) when it encounters an option letter not included in `optstring`. `EOF` is returned when processing is finished.

Author:

Hans Dietrich <hdietrich2@hotmail.com>

Bug

- Long options are not supported.
- The GNU double-colon extension is not supported.
- The environment variable `POSIXLY_CORRECT` is not supported.
- The + syntax is not supported.
- The automatic permutation of arguments is not supported.
- This implementation of getopt() returns `EOF` if an error is encountered, instead of `-1` as the latest standard requires.

Example:

```
BOOL CMyApp::ProcessCommandLine(int argc, char *argv[])
{
    int c;

    while ((c = getopt(argc, argv, "aBn:")) != EOF)
    {
        switch (c)
        {
            case 'a':
                TRACE(_T("option a\n"));
                //
                // set some flag here
                //
                break;

            case 'B':
                TRACE( _T("option B\n"));
                //
                // set some other flag here
                //
                break;

            case 'n':
                TRACE(_T("option n: value=%d\n"), atoi(optarg));
                //
                // do something with value here
                //
                break;

            case '?':
                TRACE(_T("ERROR: illegal option %s\n"), argv[optind-1]);
                return FALSE;
                break;

            default:
                TRACE(_T("WARNING: no handler for option %c\n"), c);
                return FALSE;
                break;
        }
    }
    //
    // check for non-option args here
```

```
    //
    return TRUE;
}
```

4.3.3 Variable Documentation

4.3.3.1 char* optarg

On return from getopt(), `optarg` is set to point to the start of the option argument. If there are no non-option arguments, `optarg` will be set to `NULL`.

4.3.3.2 int opterr

This version of getopt() does not fill `opterr`.

4.3.3.3 int optind = 0

getopt() places in the external variable `optind` the `argv` index of the next argument to be processed. `optind` is initialized to `0` before the first call to getopt().

4.4 tools – error handling and filename tools

4.4.1 Detailed Description

Error handling – error messages can be formatted and print with error(). The programname as set with setprogname() precedes the message. Error messages are printed to the `stderr` stream.

General messages – function message() is meant to format and print normal messages, for example to trace the program's normal processing. A message also specifies a level and the message is only printed, if its level is equal or lower than the current message verbosity level as set with setmsglevel(). Messages are printed to the stream as set with setmsgfile(), default it is `stdout`.

Debug messages – function debug() is meant to format and print debug messages, for example to trace the program's data handling. A debug message also specifies a level and the message is only printed, if its level is equal or lower than the current debug message verbosity level as set with setdbglevel(). Messages are printed to the stream as set with setdbgfile(), default it is `stdout`.

Filenames – parts of a filename path can be obtained with basename(), dirname() and extname(). Function exist() tells if the specified filename exists.

User interaction – funtion prompt() lets you request a user to supply a non-empty string. With pick() you ask a user to accept or reject its argument. ttyin() reads a single character from the terminal.

Module tools is written by Martin J. Moene, Leiden University. It was inspired by the books:

```
The Unix Programming Environment
Brian W. Kernighan and Rob Pike.
Prentice Hall, Inc., 1984.
ISBN 0-13-937681-X (paperback), 0-13-937699-2 (hardback).

The Practice of Programming.
Brian W. Kernighan and Rob Pike.
Addison-Wesley, Reading, Massachusetts, 1999.
ISBN 0-201-61586-X.
```


Defines

- #define SLASH '\\'
  filename separation slash

Enumerations

- enum { E_OK, E_OPT, E_PS, E_INT, E_ITL }
  application return codes.
- enum { MSG_L0, MSG_L1, MSG_L2 }
  application message level codes.
- enum { DBG_L0, DBG_L1, DBG_L2 }
  application debug level codes.

Functions

- const char * getprogname ()
  return the program's name as set with setprogname().
- void setprogname (const char *name)
  set the program's name as used with error() etc.
- int getmsglevel ()
  get the current message verbosity level.
- void setmsglevel (int level)
  set the message verbosity level.
- FILE * getmsgfile ()
  get the current file messages will be written to.
- void setmsgfile (FILE *file)
  set the file to write messages to.
- int getdbglevel ()
  get the current debug message verbosity level.
- void setdbglevel (int level)
  set the debug message verbosity level.
- FILE * getdbgfile ()
  get the current file debug messages are written to.
- void setdbgfile (FILE *file)
  set the file to write the debug messages to.
- int error (int status, const char *format...)
  format and print error message, return status.
- int debug (int level, const char *format...)
  format and print debug message, if level permits.
- int message (int level, const char *format...)
  format and print message, if level permits.
- const char * dirname (const char *pathname)
  return directory part of pathname.
- const char * basename (const char *pathname)
  return filename without directory part and extension.
- const char * extname (const char *pathname)
  return extension of filename.

- int exist (const char *filename)
  tell if specified filename exists.
- string prompt (const char *ps)
  prompt for non-empty string input.
- int pick (int isInteractive, const char *arg, const char *set)
  select or skip argument.
- int ttyin ()
  read a character from the terminal.

Variables

- char * progname = NULL
  program name (default not set)
- int dbglevel = 0
  current debug message level
- int msglevel = 0
  current message level
- FILE * dbgfile = stdout
  file to write debug messages to
- FILE * msgfile = stdout
  file to write messages to

4.4.2 Enumeration Type Documentation

4.4.2.1 anonymous enum

Enumeration values:

E_OK fine

E_OPT commandline option/argument error

E_PS processing error

E_INT user interrupted program

E_ITL internal/unrecognized error

4.4.2.2 anonymous enum

Enumeration values:

MSG_L0 level 0, defult on

MSG_L1 level 1

MSG_L2 level 2

4.4.2.3 anonymous enum

Enumeration values:

DBG_L0 level 0, default on

DBG_L1 level 1

DBG_L2 level 2

4.4.3 Function Documentation

4.4.3.1 const char* basename (const char * *pathname*)

basename() returns the filename part of the given path without the extension. The returned string does not contain a trailing slash.

If you specify a pathname of NULL, the program's path is used as set with setprogname().

Note:

basename() returns a static buffer, so you must use its contents before the next call to basename().

Parameters:

pathname path or NULL

Returns:

string with basename

See also:

dirname(), extname()

Examples:

main-simple.cpp.

4.4.3.2 int debug (int *level*, const char * *format*...)

debug() formats and prints the message if the specified level is equal or less than the current debug level. `format` is a printf format string, optionally followed by arguments.

The message has the following format:

[*progname*:]*message*[: *system-message*]\n

progname is included when a colon starts the format and it has been set via setprogname(); *system-message* is included when the format string ends with a colon (:).

debug() returns the specified level.

Parameters:

level this message's level

format a printf format string, optionally followed by arguments

Returns:

level

See also:

setprogname(), getdbglevel(), setdbglevel(), getdbgfile(), setdbgfile()

4.4.3.3 const char* dirname (const char * *pathname*)

dirname() returns the directory part of the given path. The returned string does not contain a trailing slash.

If you specify a pathname of NULL, the program's path is used as set with setprogname().

Note:
dirname() returns a static buffer, so you must use its contents before the next call to dirname().

Parameters:
pathname path or NULL

Returns:
string with directory part

See also:
basename(), extname()

4.4.3.4 int error (int *status*, const char * *format...*)

error() formats and prints and error message. `format` is a printf format string, optionally followed by arguments.

The error message has the following format:

[*progname*:]*message*[: *system-message*]\n

progname is included when it has been set via setprogname(); *system-message* is included when the format string ends with a colon (:).

error() returns the specified status.

Parameters:
status the error status to return, see E_OK

format a printf format string, optionally followed by arguments

Returns:
application exit status

See also:
setprogname()

Examples:
main-simple.cpp.

4.4.3.5 int exist (const char * *filename*)

exist() returns true if the specified filename is "-" (stdin) or exists as diskfile, exist() returns false otherwise.

Parameters:
filename name of file to check for

Returns:
1, 0 (exists, not exists)

See also:
prompt()

4.4.3.6 const char* extname (const char * *pathname*)

extname() returns the extension part of the given path without a dot.

If you specify a pathname of NULL, the program's path is used as set with setprogname().

Note:
extname() returns a static buffer, so you must use its contents before the next call to extname().

Parameters:
pathname path or NULL

Returns:
string with extension

See also:
dirname(), basename()

4.4.3.7 FILE* getdbgfile ()

getdbgfile() returns the current file to write the debug messages to. Default it is stdout.

Returns:
current file to write debug messages to

See also:
getdbglevel(), setdbglevel(), getdbgfile(), debug()

4.4.3.8 int getdbglevel ()

getdbglevel() returns the current level for debug messages to be printed.

Returns:
current debug verbosity level

See also:
setdbglevel(), getdbgfile(), setdbgfile(), debug()

4.4.3.9 FILE* getmsgfile ()

getmsgfile() returns the current file to write the messages to. Default it is stdout.

Returns:
current file to write messages to

See also:
setmsgfile(), getmsglevel(), setmsglevel(), message()

4.4.3.10 int getmsglevel ()

getmsglevel() returns the current level for messages to be printed.

Returns:
current verbosity level

See also:
setmsglevel(), getmsgfile(), setmsgfile(), message()

4.4.3.11 const char* getprogname ()

getprogname() returns the program's name as it has been set with setprogname(). If setprogname() has not been called, getprogname() returns NULL.

Returns:
the name of the program, or NULL

See also:
setprogname(), error(), message(), debug()

4.4.3.12 int message (int *level*, const char * *format*...)

message() formats and print the specified message if its level is equal or less than the current verbosity level as set with setmsglevel(). `format` is a printf format string, optionally followed by arguments.

The message has the following format:

[*progname*:]*message*[: *system-message*]\n

progname is included when a colon starts the format and it has been set via setprogname(); *system-message* is included when the format string ends with a colon (:).

message() returns the specified level.

Parameters:
level this message's level

format a printf format string, optionally followed by arguments

Returns:
level

See also:
setprogname(), getmsglevel(), setmsglevel(), getmsgfile(), setmsgfile()

4.4.3.13 int pick (int *isInteractive*, const char * *arg*, const char * *set*)

If `isInteractive` is true, pick() writes its argument `arg` followed by a question mark to stderr and waits for a one-key response. If the response is a member of the supplied or default `set`, pick returns 1, for all other values, it returns 0.

If `isInteractive` is false, pick() returns true immediately, without prompting.

Parameters:

isInteractive select interactively

arg argument to use or skip

set string with response characters to accept argument [yY]

Returns:

0, 1 (skip, use)

See also:

prompt()

4.4.3.14 string prompt (const char * *ps*)

prompt() issues the prompt `ps` and returns the user's response. prompt() skips leading whitespace and includes all input to the end of the line.

prompt() keeps prompting until a non-empty resonse is provided.

Parameters:

ps prompt string

Returns:

response string

See also:

pick()

4.4.3.15 void setdbgfile (FILE * *file*)

setdbgfile() sets the file to write the debug messages to. Default it is stdout.

Parameters:

file file to write debug messages to [stdout]

See also:

getdbglevel(), setdbglevel(), getdbgfile(), debug()

4.4.3.16 void setdbglevel (int *level*)

setdbglevel() sets the level for debug messages to be printed. Debug messages with a level equal or less than the current verbosity level as set with setdbglevel() will be printed. The debug level is default 0.

Parameters:

level debug verbosity level to set [0]

See also:

getdbglevel(), getdbgfile(), setdbgfile(), debug()

4.4.3.17 void setmsgfile (FILE * *file*)

setmsgfile() sets the file to write the messages to. Default it is stdout.

Parameters:
file file to write messages to [stdout]

See also:
setmsglevel(), getmsgfile(), getmsgfile(), message()

4.4.3.18 void setmsglevel (int *level*)

setmsglevel() sets the level for messages to be printed. Messages with a level equal or less than the current verbosity level as set with setmsglevel() will be printed. The message level is default 0.

Parameters:
level verbosity level to set [0]

See also:
getmsglevel(), getmsgfile(), setmsgfile(), message()

4.4.3.19 void setprogname (const char * *name*)

setprogname() sets the program's name to the one specified. It is used with error(), message() and debug().

Parameters:
name the name of the program

Returns:
none

See also:
getprogname(), error(), message(), debug()

Examples:
main-simple.cpp.

4.4.3.20 int ttyin ()

ttyin() reads a single character from the terminal, echoes and returns it.

Returns:
character read

See also:
pick()

Chapter 5

Edit Env Example Documentation

5.1 main-simple.cpp

This example shows how error() can be used.

```
#include <stdio>               // for EXIT_FAILURE
#include "tools.h"             // for setprogname(), error()

int main( int argc, char *argv[] )
{
   setprogname( basename( argv[0] ) );

   if ( argc <= 1 )
      return error( EXIT_FAILURE, "input output" );

   // ...
}
```

Chapter 6

Edit Env Page Documentation

6.1 Edit Registry via Program Regedit

Apart from editing the registry via Windows API functions, it can also be done with program `regedit`. Here is an example of how you may do that.

```
/*
 * For instructional purposes.
 *
 * Here is a version to edit the registry using program regedit.
 *
 * commandline: regedit /s file.reg
 *
 * for example, to create variable rulbus with contents isa, and remove
 * variable sublur, file.reg may contain (without the linenumbers):
 *
 * 1 REGEDIT4
 * 2
 * 3 [HKEY_CURRENT_USER\Environment]
 * 4 "rulbus"="isa"
 * 5 "sublur"=-
 */

#include "tools.h"
#include <string>
#include <process>

using namespace std;

static int writeRegEnv ( const string& name, const string& value,
                         int optcu = 0, int optrm = 0, int optkeep = 0  );

int writeRegEnv( const string& name, const string& value, int optcu /* = 0 */ )
{
   return writeRegEnv ( name, value, optcu, 0, optkeep );
}

int deleteRegEnv( const string& name, int optcu /* = 0 */ )
{
   return writeRegEnv ( name, "", optcu, 1, optkeep );
}

static int writeRegEnv ( const string& name, const string& value,
                         int optcu /* = 0 */, int optrm /* = 0 */, int optkeep /* = 0 */ )
{
   /*
```

```
     * open file for registry commands (w+b):
     */

   string regName( basename( NULL ) + string(".reg") );
   FILE *regFile = fopen( regName.c_str(), "w" );

   if ( NULL == regFile )
      return error( E_PS, "cannot open temporary file '%s' with regedit source:", regName.c_str() );

   /*
    * write registry commands in file:
    */

   fprintf( regFile,
      optcu ? "REGEDIT4\n"
              "\n"
              "[HKEY_CURRENT_USER\\Environment]\n"
            : "REGEDIT4\n"
              "\n"
              "[HKEY_LOCAL_MACHINE\\SYSTEM\\CurrentControlSet\\Control\\Session Manager\\Environment]\n"
   );

   fprintf( regFile, optrm ? "\"%s\"=-\n" : "\"%s\"=\"%s\"\n", name.c_str(), value.c_str() );

   /*
    * close file with registry commands:
    */

   if ( EOF == fclose( regFile ) )
      return error( E_PS, "cannot close temporary file '%s' with regedit source:", regName.c_str() );

   /*
    * edit registry, using: regedit /s {file}
    */

   int status = spawnlp ( P_WAIT, "regedit", "regedit", "/s", regName.c_str(), NULL );

   if      ( status < 0 ) return error( E_PS, "spawn of regedit failed:" );
   else if ( status > 0 ) return error( E_PS, "regedit returned error"   );

   /*
    * remove file with registry commands, unless option -k is specified:
    */

   if ( !optkeep && remove( regName.c_str() ) )
      return error( E_PS, "cannot remove temporary file '%s' with regedit source:", regName.c_str() );

   return E_OK;
}
```

6.2 Bug List

Member getopt(int argc, char ∗argv[], char ∗optstring)

- Long options are not supported.
- The GNU double-colon extension is not supported.
- The environment variable `POSIXLY_CORRECT` is not supported.
- The + syntax is not supported.
- The automatic permutation of arguments is not supported.
- This implementation of getopt() returns `EOF` if an error is encountered, instead of `-1` as the latest standard requires.

Index