

Version 2.8
by Lucien Cinc

µ The WOIO Package 2	
WOIO and Dynamic Data Exchange 2	
Distributing External Commands	3
Library Compatibility	3
The entry function main()	3
The exit function exit() and Errorlevels	4
External Command ARGS	4
External Command AUTOGEN	4
Control Functions	7
Screen Output Functions	8
Input Functions	15
Command Line Functions	17
Status Bar Functions	24
File Manipulation Functions	25
File Name Functions	30
Path Name Functions	34
Unix Functions	37
File Description Functions	40
Environment Functions	41
System Functions	42

tc "The WOIO Package"§**The WOIO package**

WOIO is a library that allows programs to interact (ie. perform I/O functions) with the main WinOne window. WOIO is essentially an abstract layer that sits on top of a normal windows program and provides a number of functions, that covers up just what is necessary to write a windows program. In fact a program written using WOIO, will look more like a DOS program than a Windows program. For example, WOIO programs use `main()` as the entry point, just like DOS programs do, and `printf()` will write to the main WinOne window (ie. a virtual screen), just like DOS programs write to the screen.

WOIO greatly simplifies the writing of programs intended for execution by WinOne. Those of you familiar with Windows programming would know just how much code is needed to displaying something as simple as *"hello, world"* inside a window. It takes about 2 pages of code to do this properly, registering a class, set-up up an event loop to handle events like `WM_PAINT`, creating a font, etc. Using the WOIO library the same can be accomplished with 5 lines of code :-

```
#include "woio.h"

int main(void)
{
    printf("Hello, World\n");

    return 0;
}
```

There are some additional things that need to be set-up correctly before the above program will work, for example, the compilers switches (eg. Smart Callbacks, the Memory Model, etc...) and a definition file (ie. .DEF file) are needed. All these things will be all discussed later in this document. Apart from these extra things that are needed, some programs will only need to be modified to include this library to compile. However this will NOT generally be the case.

tc "WOIO and Dynamic Data Exchange"§**WOIO and Dynamic Data Exchange**

WOIO uses a sort of backward Dynamic Data Exchange (DDE) protocol to communicate with WinOne. Essentially, functions are transformed into a message and posted to WinOne. WinOne then processes the message and posts a reply, which signals success or failure. This is the only way one program can communicate with another program and

get an instant response.

Sounds simple enough, so why is it backward?. Well, there is no initialisation or termination involved in the WinOne DDE protocol, that always occurs when using the standard DDE protocol. To understand why there is not, you have to know something about the actual problem that has to be overcome. Assume that WinOne knew nothing about a program that it needed to execute, then after executing it, WinOne would have to wait for some period of time, to see if the executing program tries to initiate a DDE conversation. This waiting time is essentially the problem. For example, how long would WinOne wait before timing out, how can you guarantee that WinOne would not time out, before a program can link via DDE to WinOne. The overhead associated with waiting for some period of time would apply to every program that is executed by WinOne, even programs that do not want to communicate with WinOne. Clearly this is not acceptable, there must be a better way to do things. There needs to be a way to determine before a program is even executed, whether or not, it wants to communicate with WinOne, so that the appropriate course of action can be taken. There is of course and here is how it is done. All windows programs contain a description string that is inserted into the program by the compiler. Description strings usually contain a short description of a program and/or the author's copyright message. This string is set in the Module Definition File (.DEF file) and is inserted into a program at compile time. Knowing this fact, it is possible to check for a predetermined string and thereby know whether or not a program will want to communicate with WinOne before it is even executed.

tc "Distributing External Commands"§**Distributing External Commands**

Users of WinOne (either registered or un-registered) may freely use the WOIO Library package to program External Commands and can distribute them in any way, shape or form. The authors of External Commands are solely responsible for the quality of the commands they program and distribute.

tc "Library Compatibility"§**Library Compatibility**

External Commands written with the WOIO Library package are all compatible upwards, that is, External Commands will still run with newer and updated versions of WinOne. However, the reverse does not apply. For example, an External Command that is written for a newer version of WinOne will not run on an older version of WinOne. The following window is displayed to inform the user that an External Command is meant for a newer version of WinOne and after pressing the OK button, the External Command will be terminated :-

tc "The entry function main()"§**The entry function main()**

WOIO programs have an entry function `main()`, which is similar to DOS programs, but with one important difference, `main()` does not include the `argc` and `argv` arguments, but instead these arguments are implemented as functions calls. The `arg...` family of functions provide a high level of functionality that the standard `argc` and `argv` arguments do not and greatly simplifies the processing of command line arguments. It is not uncommon to see programs that devote whole modules purely for handling command line arguments and this is clearly not necessary when using the WOIO library. It may be interesting to note that these functions are similar to that used internally by WinOne itself. There are 10 `arg...` functions available, which include :-

<code>argc</code>	<code>argn</code>	<code>argnstr</code>	<code>argpath</code>
<code>argv</code>	<code>args</code>	<code>argstr</code>	<code>argabs</code>
<code>argtail</code>		<code>isargstr</code>	

tc "The exit function exit() and Errorlevels"§xe "exit"§xe "errorlevel"§xe "errorlevels"§**The exit function exit() and Errorlevels**

The standard library `exit()` may be used to exit a program at any time during its execution and the value passed to it will set the errorlevel after the program exits. However, the following errorlevel values are reserved by WinOne and should not be used :-

3	- Abnormal termination (ie. ^C pressed).
255	- Incompatible WOIO library version. (ie. the External Command is meant for a newer version of WinOne).

When a program exits it should return 0 to signal a normal termination and a value greater than 0 (ie. excluding 3 and 255) to signal an error :-

```
#include "woio.h"
```

```
int main(void)
{
    printf("Hello, World\n");
}
```

```

return 0; /* normal termination */
}

```

tc "External Command ARGS"\$xe "args"\$ Command **ARGS**

Function: Displays in a tabular form the values returned by all the arg...() functions.

Syntax: **ARGS** [anything]

anything Sequence of characters. This may include strings, arguments and switches.

See Also

argc argn argnstr argpath arg_c
 argv args argstr argabs arg_v
 argtail isargstr

tc "External Command AUTOGEN"\$xe "autogen"\$ Command **AUTOGEN**

Function: Auto generate the files needed to program external commands.

Syntax: **AUTOGEN**

Note:

Command AUTOGEN generates the following files :-

Extension	Description
.CPP	C++ program file
.H	C++ header file
.RC	Resource file

.PRJ	Project file
.DSK	Disk file
.DEF	Module Definition file
CLEAN.BAT	Batch file to clean up program files

AUTOGEN asks the following questions in order to generate the correct files :-

Question	Description
Command Name	Specifies the name of the External Command.
Command Author	Specifies the author of the External Command. This question is optional. Press return to skip the question.
Memory Model (S\M\C\L)	Specifies the memory model to use. Enter 'S' to specify the Small Model, enter 'M' to specify the Medium Model, enter 'C' to specify the Compact Model and enter 'L' to specify the Large Model.
Compiler Path [D:\TCWIN]	Specifies the directory in which the compiler is located. The default path is D:\TCWIN

All the files are generated in the current directory and AUTOGEN assumes the library files (ie. WOIO.H, WOIOS.LIB, WOIOM.LIB, WOIOC.LIB and BWCC.LIB) needed to program external commands are also located in the current directory.

Borland or Turbo C/C++ Compilers for Windows

The WOIO package is compiled using Borland C/C++ for windows. Therefore, to use the WOIO package you will need to use either the Borland or Turbo C/C++ compilers for Windows 3.0 and above.

Memory Model

There are four libraries supplied with the WOIO package that support all the memory models, as follows :-

WOIOS.LIB	supports the Small Memory model.
WOIOM.LIB	supports the Medium Memory model.
WOIOC.LIB	supports the Compact Memory Model.
WOIOL.LIB	supports the Large Memory Model .

The Module Definition file (.DEF)

A module definition file (ie. .DEF file) needs to be created (usually by command AUTOGEN), so that WinOne can recognise external commands and take the appropriate course of action. WinOne uses the DESCRIPTION field in the .DEF file to determine whether a program is an external command. For example, consider the following sample .DEF file :-

```
NAME      XXXX
DESCRIPTION 'WinOne Command XXXX - Copyright etc...'
EXETYPE   WINDOWS
CODE      PRELOAD MOVEABLE
DATA      PRELOAD MOVEABLE MULTIPLE
HEAPSIZE  37888
STACKSIZE 6144
```

The DESCRIPTION field must start with 'WinOne Command'. This string is case sensitive. The 'XXXX' part of this field is not currently checked, but will be in latter releases of WinOne, so please include it, in capital letters. The rest of the string is ignored. Insert the .DEF file into your project file and this is all that needs to be done, so that WinOne can determine whether or not your program is an external command and therefore, need special processing.

Stdin, Stdout and Stderr

The standard I/O streams `stdin`, `stdout` and `stderr` are not supported by the WOIO library. Instead WOIO provides a number of functions that attempt to simulate these streams. Functions that requires `stdin`, `stdout` or `stderr` to be past as a parameter will NOT work, and should NOT be used. For example, `fputc(c, stdout)`, `putc(c, stdout)`, `fgetc(stdin)` or `getc(stdin)` should not be used. When a program uses these functions, the program will still compile and run, but will not produce the expected result. You may very well be wondering what happens when a program uses these functions, well, the output will most likely end up writing over the desktop window. WOIO supports the following functions, which attempt to mimic the standard I/O functions of the same name as closely as is possible :-

<code>printf</code>	<code>textcolor</code>	<code>scrwidth</code>	<code>scanf</code>
<code>puts</code>	<code>clrscr</code>	<code>scrheight</code>	<code>gets</code>
<code>putchar</code>		<code>clreol</code>	<code>getchar</code>
<code>putch</code>		<code>gotoxy</code>	<code>getch</code>
<code>perror</code>		<code>wherex</code>	
		<code>wherey</code>	

Actually, `stdin`, `stdout` and `stderr` is a special case of a much larger problem. In general, it is not recommended to use any functions from the standard I/O library that uses buffered I/O or file streams (ie. `FILE *stream`). For example `fopen`, `fread`, `fwrite`, `fclose`, `fprintf`, etc..., should not be used, instead un-buffered file I/O should be used, for example `open`, `close`, `read`, `write`, `_dos_open`, `_dos_close`, `_dos_read`, `_dos_write`, `OpenFile`, etc... Un-buffered file I/O will allow yield points to be inserted into the code, so that other tasks can run (ie. multi task).

tc "Control Functions"§ **Control Functions**

tc "Function SCREEN"§xe "screen"§**void screen(flag)**

int flag /* BUFFERED or UNBUFFERED value */

Enable or disable buffered screen output. When writing to the screen the text will not appear until enough lines have been written to fill one complete screen.

Parameter	Description
flag	When this value is BUFFERED, then the screen will only update after each screen full. When this value is UNBUFFERED, then the screen is updated to show any lines not yet displayed.

Returns

There is no return value.

Comments

When screen() is set to BUFFERED it should be set to UNBUFFERED before the program terminates. When screen() is not used then the screen is updated as it is written to.

The values BUFFERED and UNBUFFERED are #defined in the WOIO.H file.

Example

```
#include "woio.h"

int main(void)
{
    int ret;

    screen(BUFFERED); /* enable buffered output */

    ret = dofunction();

    screen(UNBUFFERED); /* flush any lines not displayed */

    return ret; /* error level */
}
```

tc "Function YIELD"§xe "yield"§**void yield(void)**

Allows other tasks that are running to multitask.

Returns

There is no return value.

Comments

Since Windows is a co-operative multitasking operating system, yield() should be called when a long period of time elapses without calling any of the WOIO library functions.

Example

```
#include "woio.h"

int main(void)
{
    printf("press ^C to quit: \n");

    while (1)
        yield();          /* allow other tasks to multitask */
}
```

tc "Screen Output Functions"§**Screen Output Functions**

tc "Function PRINTF"§xe "printf"§**int printf(fmt, ...)**

printf() provides formatted output and functions similar to the standard run time library printf().

Comments

For a full description of printf() consult your standard run time library reference manual.

printf() will only display printable characters.

The following #define values can be past as character arguments to change the colour of the text displayed by printf() :-

```
#define BLACK          (char )128 /* text colours */
#define RED            (char )129
#define GREEN          (char )130
#define BLUE           (char )131
#define YELLOW         (char )132
#define MAGENTA        (char )133
```

```

#define CYAN          (char )134
#define WHITE        (char )135
#define LIGHTGRAY    (char )136
#define LIGHTRED     (char )137
#define LIGHTGREEN   (char )138
#define LIGHTBLUE    (char )139
#define BROWN        (char )140
#define LIGHTMAGENTA (char )141
#define LIGHTCYAN    (char )142
#define DARKGRAY     (char )143

```

See Also

- puts
- putch
- putchar
- textcolor
- scanf
- gets

Example

Consider the following :-

```
printf("%cHello, world\n%cHow are you?", MAGENTA, BLUE);
```

will display the following :-

```

Hello, world
How are you?

```

tc "Function PUTS"§xe "puts"§**void puts(s)**

char *s /* character string */

Display a character string along with a CR-LF character combination.

Parameter	Description
s	Address of a NULL terminated character string.

Returns

There is no return value.

Comments

Function puts() is streamable, that is, when stdout is redirected on the command line to a file, the character string will be written to the file.

Tab characters are padded with space characters.

See Also

- printf
- putchar
- putch
- gets

tc "Function PUTCH"§xe "putch"§**void putch(c)**

char c /* character */

Display a character directly to the screen.

Parameter	Description
c	Character value.

Returns

There is no return value.

Comments

Function putch() writes directly to the screen, and as a result is not streamable.

Tab characters are padded with space characters.

See Also

- printf
- puts
- putchar
- getch

Example

```
void myerror(char *msg)
```

```

{
    textcolor(RED);    /* display message in RED */

    while (*msg)      /* display message */
        putchar(*msg++);

    putchar('\n');
}

```

tc "Function PUTCHAR"\$xe "putchar"\$**void putchar(c)**

char c /* character */

Display a character.

Parameter	Description
c	Character value.

Returns

There is no return value.

Comments

Function putchar() is streamable, that is, when stdout is redirected on the command line to a file, the character will be written to the file.

Tab characters are padded with space characters.

See Also

printf
puts
putch
getchar

Example

```

void charmsg(char *msg)
{
    while (*msg)      /* display message */
        putchar(*msg++);
}

```

```
    putchar('\n');
}
```

tc "Global variable ERRCNT"\$xe "errcnt"\$**int errcnt**

Global variables that is incremented by one each time perror() is called to display an error message.

tc "Function PERROR"\$xe "perror"\$ **void perror(msg)**

char *msg /* character string */

Display an error message.

Parameter	Description
msg	Address of a NULL terminated character string that contains the message to display.

Returns

There is no return value.

Comments

The message along with two CR-LF character combinations is written to stderr. Stderr is not streamable, that is, when stdout is redirected on the command line to a file, stderr will still write to the screen.

RED is used for the foreground colour.

See Also

- errcnt
- printf
- puts
- putchar

tc "Function TEXTCOLOR"\$xe "textcolor"\$void **textcolor(col)**

char col /* colour value */

Set the current text colour.

Parameter	Description
col	Range value, that specifies the colour to set.

Returns

There is no return value.

Comments

The following colours are #defined in the WOIO.H header file :-

```
#define BLACK                (char )128  /* text colours */
#define RED                  (char )129
#define GREEN                (char )130
#define BLUE                (char )131
#define YELLOW               (char )132
#define MAGENTA              (char )133
#define CYAN                 (char )134
#define WHITE                (char )135
#define LIGHTGRAY            (char )136
#define LIGHTRED             (char )137
#define LIGHTGREEN           (char )138
#define LIGHTBLUE            (char )139
#define BROWN                (char )140
#define LIGHTMAGENTA        (char )141
#define LIGHTCYAN            (char )142
#define DARKGRAY             (char )143
```

There are no blinking or bold characters.

See Also

printf

tc "Function CLRSCR"§xe "clrscr"§**void clrscr(void)**

Clear the screen.

Returns

There is no return value

Comments

When the screen is cleared, the contents of the screen are NOT moved to the scroll back buffer.

tc "Function CLREOL"§xe "clreol"§**void clreol(void)**

Clear till the end of the current line.

Returns

There is no return value

tc "Function GOTOXY"§xe "gotoxy"§**void gotoxy(x, y)**

int x /* co-ordinate */

int y /* co-ordinate */

Position the caret on the screen.

Parameter	Description
x	Co-ordinate on the horizontal x-axis.
y	Co-ordinate on the vertical y-axis.

Returns

There is no return value

Comments

The first character on the screen is at co-ordinate 1, 1.

See Also

wherex
wherey

Example

```
#include "woio.h"
#include <string.h>

/*
   Display the string "Hello, World"
   centred on the screen
*/

int main(void)
{
    char *s;

    s = "Hello, World"; /* string to display */
    clrscr();

    gotoxy((scrwidth() - strlen(s)) / 2, scrheight() / 2);
    printf("%c%s", WHITE, s);

    gotoxy(0, scrheight());

    return 0; /* error level */
}
```

tc "Function WHEREX"§xe "wherex"§**int wherex(void)**

Determine the horizontal location of the caret.

Returns

Co-ordinate of the caret on the horizontal x-axis.

Comments

Co-ordinates start from 1.

See Also

wherey
gotoxy

tc "Function WHEREY"\$xe "wherey"\$int **wherey(void)**

Determine the vertical location of the caret.

Returns

Co-ordinate of the caret on the vertical y-axis.

Comments

Co-ordinate start from 1.

See Also

wherex
gotoxy

tc "Function SCRWIDTH"\$xe "scrwidth"\$xe "screenwidth"\$int **scrwidth(void)**

Determine the screen width in characters.

Returns

The screen width in characters.

See Also

scrheight

tc "Function SCRHEIGHT"\$xe "scrheight"\$xe "screenheight"\$int **scrheight(void)**

Determine the screen height in characters.

Returns

The screen height in characters.

See Also

scrwidth

tc "Function ATOC"§xe "atoc"§xe "comma"§xe "commas"§**char *atoc(number)**

char *number /* character string */

Delimit a character string containing a number with comma's, every thousand.

Parameter	Description
number	Address of a the character string containing the number to convert.

Returns

The address of the original character string number.

Comments

The character string number must be large enough to hold the comma's inserted into it.

tc "Input Functions"§**Input Functions**

tc "Function SCANF"§xe "scanf"§**int scanf(fmt, ...)**

scanf() provides formatted input and functions the same as the standard run time library scanf().

Comments

For a full description of scanf() consult your standard run time library reference manual.

See Also

gets
getch
getchar
printf

tc "Function GETS"§xe "gets"§**char *gets(s)**

char *s /* character string */

Get a character string without the CR-LF character combination.

Parameter	Description
s	Address of a character array to store the string. This array must be at least 80 characters in size.

Returns

On success, it returns the address of the character array, where the NULL terminated character string is stored. On end of file (ie. EOF) or on error, NULL is returned.

Comments

Function gets() is streamable, that is, when stdin is redirected on the command line from a file, the character string will be read from the file and will not be echoed to the screen.

Tab characters are converted to a single space character, unless stdin has been redirected on the command line.

See Also

scanf
getch
getchar
puts

Example

```
#include "woio.h"  
#include <dos.h>
```

```
/* Determine whether a file exists */
```

```
int prompt_open(void)
```

```

{
    char buf[80];
    int handle;

    printf("%cEnter filename:%c ", WHITE, LIGHTGRAY);

    if (gets(buf) /* get a filename */
        if (_dos_open(buf, 0, &handle)
            return handle; /* opened file */

    return 0; /* failed to open file */
}

int main(void)
{
    int handle;

    if ((handle = prompt_open()) != 0) {
        printf("File exists\n");
        _dos_close(handle); /* close the file */
        return 1;
    }

    return 0;
}

```

tc "Function GETCHAR"\$xe "getchar"\$i**int getchar(void)**

Get a character .

Returns

On success, a character value is returned, on error or end of file, a value of EOF (ie. -1) is returned.

Comments

Function `getchar()` is streamable, that is, when `stdin` is redirected on the command line from a file, the characters will be read from the file and will not be echoed to the screen.

When `stdin` has NOT been redirected on the command line then the following applies :-

1. Characters are echoed to the screen.
2. Tab characters are converted to single space characters,
3. Carriage return characters are converted to new line characters (ie. '\r' mapped to '\n').
4. All non-printable characters are ignored.

See Also

- scanf
- getch
- putch
- putchar

tc "Function GETCH"\$xe "getch"\$§**int getch(void)**

Get a character from the keyboard

Returns

A character value.

Comments

Function getch() read characters from the keyboard, and as a result is not streamable.

Characters read are not echoed to the screen.

There is no character mapping. (ie. '\r' is NOT mapped to '\n');

See Also

- scanf
- putch
- getchar
- putchar

tc "Command Line Functions"\$§**Command Line Functions**

tc "Global variables _ARGC and _ARGV"\$xe "_argc"\$xe "_argv"\$§**int arg_c**

char *arg_v[]

Global variables that contains the number of command line arguments (ie. arg_c) and the

actual command line arguments (ie. `arg_v`).

Comments

`arg_c` and `arg_v` is provided for compatibility with the standard library `argc` and `argv`, which is past to a normal C or C++ `main()`, and has the following format :-

```
#include "stdio.h"

int main(int argc, char *argv[])
{
}
```

and a WinOne external command `main()`, has the following format :-

```
#include "woio.h"

int main(void)
{
    /* arg_c is used instead of argc */
    /* arg_v is used instead of argv */
}
```

When using `arg_c` and `arg_v`, avoid using the `arg...()` family of functions, since `arg_c` and `arg_v` do not separate command line arguments and command line switches.

See Also

- `argc`
- `argv`
- `argn`
- `args`
- `argstr`
- `argnstr`

tc "Function ARGV"§xe "argc"§int **argc(void)**

Determines the number of command line arguments.

Returns

The number of command line arguments.

Comments

Command line strings (eg. "This is a string") are considered as command line arguments. Command line switches are not considered as part of the command line arguments.

See Also

- `argv`

argn
args
argstr
argnstr

tc "Function ARGV"§xe "argv"§char *argv(index)

int index /* command line argument */

Retrieve a command line argument.

Parameter	Description
index	Specifies which argument to retrieve. Specifying an index of 0 retrieves the programs name. Command line arguments start from an index of 1.

Returns

On success it returns the address of a NULL terminated string containing the argument. On error it returns a NULL.

Comments

The argument is stored in a static buffer and is over-written each time this function is called. This function cannot be used to retrieve command line switches.

See Also

argc
argn
args
argstr
argnstr

Example

```
#include "woio.h"  
#include <stdlib.h>
```

```
/* Sum all value on the command line */
```

```
int main(void)  
{  
    long total;
```



```

int i, n;

total = 0; /* zero total */
if ((n = argc()) == 0) {
    perror("nothing to sum");
    return 1;
}

for (i = 0; i < n; i++)

    total += atol(argv(i + 1));

printf("%ctotal=%c%ld\n", WHITE, YELLOW, total);

return 0; /* error level */
}

```

tc "Function ARGPATH"§xe "argpath"§**char *argpath(index)**

int index /* command line argument */

Retrieve a command line argument and convert it to a full path name.

Parameter	Description
index	Specifies which argument to retrieve. Specifying an index of 0 retrieves the current directory, as a full path name. Command line arguments start from an index of 1.

Returns

On success it returns a the address of a NULL terminated string containing the full path name. On error it returns a NULL.

Comments

The full path name is stored in a static buffer and is over-written each time this function is called.

Full path names are made up of the following components :-

drive:\directory\filename

Component	When not Specified
drive	Current drive is used.
directory	Current directory is used. Also relative directories are converted to absolute directories.
filename	*.* is used. Wildcard characters are allowed in the filename.

See Also

argc
argv
argabs

Examples

The following examples assume the current directory is C:\WINDOWS :-

Argument	Full path
C:	C:\WINDOWS*.*
C:\	C:*.*
\DOS\	C:\DOS*.*
NOTEPAD.EXE	C:\WINDOWS\notepad.exe
.EXE	C:\WINDOWS.exe
WHAT	C:\WINDOWS\what.
.	C:\WINDOWS*.*
.	C:\WINDOWS*.*

..

C:*.*

tc "Function ARGABS"§xe "argabs"§**char *argabs(index)**

int index /* command line argument */

Retrieve a command line argument and convert it to an absolute path name.

Parameter	Description
index	Specifies which argument to retrieve. Specifying an index of 0 retrieves the current directory, as an absolute path name. Command line arguments start from an index of 1.

Returns

On success it returns the address of a NULL terminated string containing the absolute path name. On error it returns a NULL.

Comments

The absolute path name is stored in a static buffer and is over-written each time this function is called.

Absolute path names are made up of the following components :-

drive:\directory\filename

Component	When not Specified
drive	Current drive is used.
directory	Current directory is used. Also relative directories are converted to absolute directories.
filename	The previous directory name becomes the



filename. Wildcard characters are allowed in filename.

See Also

- argc
- argv
- argpath

Example

The following examples assume the current directory is C:\WINDOWS :-

Argument	Full path
C:	C:\WINDOWS.
C:\	C:\.
\DOS\	C:\DOS.
NOTEPAD.EXE	C:\WINDOWS\NOTEPAD.EXE
.EXE	C:\WINDOWS.EXE
WHAT	C:\WINDOWS\WHAT.
.	C:\WINDOWS*.*
.	C:\WINDOWS.
..	C:\.

tc "Function ARGTAIL"§xe "argtail"§**char *argtail(void)**

Retrieve the actual command line tail.

Returns

Address of a NULL terminated string containing the command line tail.

Comments

The actual command line tail does not include any redirection arguments. Also, the tail is stored in a static buffer and is over-written each time this function is called.

See Also

- argc
- argv
- argn
- args

tc "Function ARGN"§xe "argn"§**int argn(void)**

Determines the number of command line switches.

Returns

The number of switches.

See Also

- argc
- argv
- args

tc "Function ARGS"§xe "args"§**char *args(void)**

Retrieve the command line switches.

Returns

Address of a NULL terminated string containing all the switches.

Comments

Switches are stored in a static buffer and is over-written each time this function is called.

The string returned can be empty, when there are no command line switches.

See Also

argc
argv
argn

tc "Function ARGSTR"§xe "argstr"§**char *argstr(index)**

int index /* command line argument */

Retrieve a command line argument and convert it from a string.

Parameter	Description
index	Specifies which argument to retrieve. Command line arguments start from an index of 1.

Returns

On success it returns the address of a NULL terminated string containing the converted command line string. On error it returns a NULL.

Comments

Strings are converted by removing the first and last double-quote marks characters, and by replacing any pair of double-quote marks that appear inside the string with a single double-quote mark character.

This function converts the actual command line argument and then returns it. After a command line argument is converted it is no longer considered a command line string.

The converted command line string is stored in a static buffer and is over-written each time this function is called.

See Also

argc
argv
argnstr
isargstr

Example

Converting the following string "This is a ""string""." returns This is a "string".

```
tc "Function ARGNSTR"§xe "argnstr"§int argnstr(void)
```

Determines the number of command line arguments, that can be converted from a string.

Returns

The number of command line arguments that can be converted from a string.

See Also

- argc
- argv
- argstr
- isargstr

```
tc "Function ISARGSTR"§xe "isargstr"§BOOL isargstr(index)
```

```
int index          /* command line argument */
```

Determines whether a command line argument can be converted from a string.

Parameter	Description
index	Specifies which argument to check. Command line arguments start from an index of 1.

Returns

On success, a non-zero value when a command line argument can be converted from a string and zero otherwise.

See Also

- argc
- argv
- argnstr
- argstr

tc "Status Bar Functions"§**Status Bar Functions**

tc "Function LIMIT"§xe "limit"§**void limit(upper)**

unsigned long upper /* upper limit */

Set the Status Bar upper limit (ie. target value) to reach.

Parameter	Description
upper	Specifies the upper limit to reach.

Returns

There is no return value.

Comments

This will display 0 in the Precent indicator, the next time the display is updated.

See Also

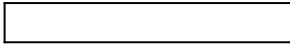
inc
empty

tc "Function INC"§xe "inc"§**void inc(value)**

unsigned long value /* value to increment by */

Increment the current Status Bar total.

Parameter	Description
value	Specifies a value to be added to the current Status Bar total.

**Returns**

There is no return value.

Comments

Function inc() may be called many times before a percentage is calculated and displayed, since the Status Bar is updated once every second.

See Also

limit
empty

tc "Function EMPTY"§xe "empty"§**void empty(void)**

Clear the Status Bar Percent indicator.

Returns

There is no return value.

Comments

The Status Bar Percent indicator is blanked unconditionally.

See Also

limit
inc

tc "File Manipulation Functions"§ **File Manipulation Functions**

tc "Function FILESIZE"§xe "filesize"§**long filesize(path)**

char *path /* file name path */

Retrieve the size of a file.

Parameter	Description
path	Address of a NULL terminated character string containing the path of a filename. This path can be relative or absolute and can not contain wildcard characters.

Returns

The size (in bytes) of the specified file. When the file does not exist -1 is returned.

Comments

On error filesize() does NOT display an error message.

See Also

filecpy
filencpy
filecat
filencat

tc "Function FILECPY"\$xe "filecpy"\$long **filecpy(dst, src, o_flag, u_flag)**

char *dst /* file name path */
char *src /* file name path */
int o_flag /* open file flags */
int u_flag /* update status bar flags */

Copies the source file to the destination file.

Parameter	Description
dst	Address of a NULL terminated character string containing the path of the destination file. This path can be relative or absolute and can NOT contain wildcard characters.
src	Address of a NULL terminated character string containing the path of the source file. This path can be relative or absolute and can NOT contain wildcard characters.
o_flag	Value which specifies how to open the destination file. These values are #defined in the WOIO.H header file and include:- <div style="text-align: center;"> O_OPEN Open the destination file. </div>

	<p>O_CREATE Create the destination file. When the destination file already exists then it is truncated to 0 length.</p> <p>O_CREATNEW Create the destination file. When the destination file already exists then filecopy() will fail.</p>
u_flag	<p>Value which specifies how to update the status bar percentage. These values are #defined in the WOIO.H header file and include:-</p> <p>INC_BYTE Increment by the size of the source file.</p> <p>INC_ONE Increment by a value of 1 only.</p> <p>INC_NONE Do not update.</p>

These values can NOT be combined.

Returns

On success, it returns the number of bytes copied to the destination file. On error a value of -1 is returned.

Comments

On error, filecopy() displays one of the following error messages:-

- Out of memory
- Invalid path or file name
- Destination file already exists
- Read error
- Insufficient disk space

See Also

- filesize
- filecopy
- filecat
- filencat

tc "Function FILECAT"\$xe "filecat"\$s**long filecat(dst, src, u_flag)**

```
char *dst          /* file name path */
char *src          /* file name path */
int u_flag        /* update status bar flags */
```

Appends the source file to the end of the destination file.

Parameter	Description
dst	Address of a NULL terminated character string containing the path of the destination file. This path can be relative or absolute and can NOT contain wildcard characters.
src	Address of a NULL terminated character string containing the path of the source file. This path can be relative or absolute and can NOT contain wildcard characters.
u_flag	<p>Value which specifies how to update the status bar percentage. These values are #defined in the WOIO.H header file and include:-</p> <p style="text-align: center;"> INC_BYTE Increment by the size of the source file. INC_ONE Increment by a value of 1 only. INC_NONE Do not update. </p>

Returns

On success, it returns the number of bytes appended to the destination file. On error a value of -1 is returned.

Comments

On error, filecat() displays one of the following error messages:-

Out of memory
Invalid path or file name
Destination file already exists
Seek error
Read error
Insufficient disk space

See Also

filesize
filecpy
filencpy
filencat

tc "Function FILENCOPY"§xe "filencpy"§**long filencpy(dst, src, num, offset, o_flag, u_flag)**

char *dst /* file name path */
char *src /* file name path */
long num /* number of bytes to copy */
long offset /* starting offset */
int o_flag /* open file flags */
int u_flag /* update status bar flags */

Copy a part of the source file to the destination file.

Parameter	Description
dst	Address of a NULL terminated character string containing the path of the destination file. This path can be relative or absolute and can NOT contain wildcard characters.
src	Address of a NULL terminated character string containing the path of the source file. This path can be relative or absolute and can NOT contain wildcard characters.
num	Value specifying the number of bytes in the source file that should be copied to the destination file. A value of -1 indicates that all the remaining bytes in the source file, taking into account the starting offset, should be copied.
offset	Value specifying an offset into the source file, where characters will be read from, The first byte in the source file is located at offset zero.
o_flag	Value which specifies how to open the destination file. These values are #defined in the WOIO.H header file and include:- <ul style="list-style-type: none"> O_OPEN Open the destination file. O_CREATE Create the destination file. When the destination file already exists then it is truncated to 0 length. O_CREATNEW Create the destination file. When the destination file already exists then filencpy() will fail.

u_flag

These values can NOT be combined.

Value which specifies how to update the status bar percentage. These values are #defined in the WOIO.H header file and include:-

```

INC_BYTE   Increment by the size of the source file.
INC_ONE    Increment by a value of 1 only.
INC_NONE   Do not update.

```

Returns

On success, it returns the number of bytes copied to the destination file. On error a value of -1 is returned.

Comments

On error, filencpy() displays one of the following error messages:-

```

Out of memory
Invalid path or file name
Destination file already exists
Out of range
Seek error
Read error
Insufficient disk space

```

See Also

```

filesize
filecpy
filecat
filencat

```

tc "Function FILENCAT" \$xe "filencat" § **long filencat(dst, src, num, offset, u_flag)**

```

char *dst           /* file name path */
char *src           /* file name path */
long num            /* number of bytes to copy */
long offset         /* starting offset */
int u_flag          /* update status bar flags */

```

Appends a part of the source file to the end of the destination file.

Parameter	Description
-----------	-------------

dst	Address of a NULL terminated character string containing the path of the destination file. This path can be relative or absolute and can NOT contain wildcard characters.
src	Address of a NULL terminated character string containing the path of the source file. This path can be relative or absolute and can NOT contain wildcard characters.
num	Value specifying the number of bytes in the source file that should be appended to the destination file. A value of -1 indicates that all the remaining bytes in the source file, taking into account the starting offset, should be appended.
offset	Value specifying an offset into the source file, where characters will be read from, The first byte in the source file is located at offset zero.
u_flag	Value which specifies how to update the status bar percentage. These values are #defined in the WOIO.H header file and include:- <p style="text-align: center;"> INC_BYTE Increment by the size of the source file. INC_ONE Increment by a value of 1 only. INC_NONE Do not update. </p>

Returns

On success, it returns the number of bytes appended to the destination file. On error a value of -1 is returned.

Comments

On error, filecat() displays one of the following error messages:-

Out of memory
Invalid path or file name
Destination file already exists
Out of range
Seek error
Read error
Insufficient disk space

See Also

filesize
filecpy
filencpy
filecat

tc "File Name Functions"§**File Name Functions**

tc "Function FILLFILE"§xe "fillfile"§**int fillfile(path, attr)**

char *path /* directory path */
unsigned int attr /* file attributes */

Create a File Table of all the files in a given path and with a given attribute.

Parameter	Description
path	Address of a NULL terminated character string containing a path. This path can be relative or absolute and can contain Wildcard characters any where in the filename part of the path.
attr	DOS File attribute (defined in DOS.H), include the following :- FA_RDONLY Read-only FA_HIDDEN Hidden file FA_SYSTEM System file FA_LABEL Volume label FA_DIREC Directory FA_ARCH Archive

Returns

On success, it returns the number of files that match the path and attribute specified, otherwise, it returns a value of zero, when no files match.

Comments

Function fillfile() replaces findfirst(), findnext(), _dos_findfirst() and _dos_findnext(), since these functions do not support extended wildcard card characters.

WinOne allows wildcard characters to be placed anywhere inside a filename and be correctly interpreted.

All files that meet the specifies requirements are places inside a table (ie. File Table) which is over-written with each call to this function. Use the getfile...() functions to

access the information stored in this table.

The table is sorted in alphabetical order.

See Also

getfile
getfilepath
getfilename

tc "Function GETFILE"§xe "getfile"§**BOOL getfile(index, pff)**

int index /* index into File Table */
struct fblk *pff /* DOS file control block structure */

Retrieve a DOS file control block structure from the File Table.

Parameter	Description
index	Specifies which fblk to retrieve from the File Table. Entries in the File Table start from an index of 0.
pff	Address of a DOS file control block structure (defined in DIR.H) :- struct fblk { char ff_reserved[21]; /* reserved by DOS */ char ff_attrib; /* attribute found */ int ffftime; /* file time */ int ff_fdate; /* file date */ long ff_fsize; /* file size */ char ff_name[13]; /* found file name */ };

Returns

On success, it returns a non-zero value and the file control block structure is filled. On error zero is returned.

Comments

Use the fillfile() to fill the File Table before using getfile().

See Also

fillfile

```
getfilepath
getfilename
```

Example

```
#include "woio.h"
#include <dir.h>

/* Display a file listing */

int dir(char *path)
{
    int i, n;
    struct fblk fblk;

    printf("%cDirectory of %s\n\n", WHITE, path);

    if ((n = fillfile(path, 0)) == 0) {
        perror("No files found");
        return 1;
    }

    for (i = 0; i < n; i++) {

        if (getfile(i, &fblk) == FALSE) {
            perror("Bad index");
            return 1;
        }

        printf(" %c%-13s%c%9ld\n",
            GREEN, fblk.ff_name,
            YELLOW, fblk.ff_fsize); /* display file names */

    }

    return 0; /* all done */
}

int main(void)
{
    char *path;
    int ret;

    if (argc() > 1) { /* check number of arguments */
        perror("Too many or few arguments");
        return 1;
    }
}
```

```

screen(BUFFERED); /* buffered screen output */

if ((path = argpath(argc)) == NULL) {
    perror("Path or file not found");
    return 1;
}

ret = dir(path);

screen(UNBUFFERED); /* flush output */

return ret; /* error level */
}

tc "Function GETFILEPATH"§xe "getfilepath"§char *getfilepath(index)

```

int index /* index into File Table */

Retrieve a file path from the File Table.

Parameter	Description
index	Specifies which file path to retrieve from the File Table. Entries in the File Table start from an index of 0.

Returns

On success it returns the address of a NULL terminated string containing the path name. On error it returns a NULL.

Comments

Use the fillfile() to fill the File Table before using getfilepath().

Path names are stored in a static buffer and is over-written each time this function is called.

The path returned may not contain a fully qualified path name.

See Also

- fillfile
- getfile
- getfilename

tc "Function GETFILENAME"%xe "getfilename"%xchar ***getfilename(index)**

int index /* index into File Table */

Retrieve a file name from the File Table.

Parameter	Description
index	Specifies which file name to retrieve from the File Table. Entries in the File Table start from an index of 0.

Returns

On success it returns the address of a NULL terminated string containing the file name. On error it returns a NULL.

Comments

Use the fillfile() to fill the File Table before using getfilename().

File names are stored in a static buffer and is over-written each time this function is called.

See Also

fillfile
getfile
getfilepath

tc "Function PADFILENAME"%xe "padfilename"%xe "pad"%xchar ***padfilename(path)**

char *path /* character string */

Pad a file name so that it is suitable for displaying.

Parameter	Description
path	Address of a NULL terminated character string contain a path name.

Returns

The address of a character string containing the padded file name.

Comments

Only the file name is padded and returned, the rest of the path is discarded.

The padded file name is stored in a static buffer and is over-written each time this function is called.

tc "Path Name Functions"§**Path Name Functions**

tc "Function FILLPATH"§xe "fillpath"§**int fillpath(path)**

char *path /* directory path */

Create a Path Table containing all the directories and sub-directories starting from the specified path.

Parameter	Description
path	Address of a NULL terminated character string containing a path. This path can be relative or absolute. The file name part of the path is expected and ignored.

Returns

On success, it returns the number of directories and sub-directories found, otherwise, it returns a value of zero, when no directories are found.

Comments

The directories . and .. are not included in the table.

A new Path Table is allocated with each call to this function. Use freepaths() to release the memory allocated for the table, when it is no longer needed.

The table is sorted in alphabetical order

See Also

fillpathall
freepaths
getpath

tc "Function FILLPATHALL"§xe "fillpathall"§**int fillpathall(void)**

Create a Path Table containing all the directories and sub-directories for all drives that are non-removable.

Returns

On success, it returns the number of directories and sub-directories found, otherwise, it returns a value of zero, when no directories are found.

Comments

The directories . and .. are not included in the table.

A new Path Table is allocated with each call to this function. Use freepaths() to release the memory allocated for the table, when it is no longer needed.

The table is sorted in alphabetical order

See Also

fillpath
freepaths
getpath

tc "Function FREEPATHS"§xe "freepaths"§**void freepaths(void)**

Release the memory allocated to store the Path Table.

Returns

There is no return value.

See Also

fillpath
fillpathall

tc "Function GETPATH"§xe "getpath"§**char *getpath(index)**

int index /* index into the Path Table */

Retrieve a path name from the Path Table.

Parameter	Description
index	Specifies which path to retrieve from the Path Table. Entries in the Path Table start from an index of 0.

Returns

On success it returns the address of a NULL terminated string containing the path name. On error it returns a NULL.

Comments

All the path names returned contains *.* for the file name part..

Use the fillpath() or fillpathall() to fill the Path Table before using getpath().

Path names are stored in a static buffer and is over-written each time this function is called.

See Also

fillpath
fillpathall
freepaths

tc "Function TOTALFILES"§xe "totalfiles"§xe "totalfile"§**int totalfiles(void)**

Get the total number of bytes for all the files in the directories contained in the Path Table.

Returns

The total number of bytes for all the files in the Path Table.

Comments

Use the fillpath() or fillpathall() to fill the Path Table before using totalfiles().

The total number of bytes includes all files, irrespective of the file attributes.

See Also

fillpath
fillpathall

tc "Unix Functions"§**Unix Functions**

tc "Function UNIXPATH"§xe "unixpath"§**char *unixpath(path)**

char *path /* DOS path to convert */

Convert a DOS path name to a Unix path name, when Unix mode is enabled.

Parameter	Description
path	Address of a NULL terminated character string containing a DOS path. This path can be relative or absolute.

Returns

The address of a NULL terminated string containing the Unix path name, when Unix mode is enabled, otherwise, the DOS path name is returned.

Comments

Path names are stored in a static buffer and is over-written each time this function is called.

When Unix mode is not enabled, then the DOS path name is simply copied into the static buffer and is not converted.

See Also

unixcmd
isunix
todopath
todoscnd

tc "Function UNIXCMD"§xe "unixcmd"§**char *unixcmd(cmd)**

char *cmd /* DOS command to convert */

Convert a DOS command line to a Unix command line, when Unix mode is enabled.

Parameter	Description
cmd	Address of a NULL terminated character string containing a DOS command line.

Returns

The address of a NULL terminated string containing the Unix command line, when Unix mode is enabled, otherwise, the DOS command line is returned.

Comments

Command lines are stored in a static buffer and is over-written each time this function is called.

When Unix mode is not enabled, then the DOS command line is simply copied into the static buffer and is not converted.

See Also

- unixpath
- isunix
- todospath
- todoscmd

tc "Function ISUNIX"§xe "isunix"§**int isunix(void)**

Determines whether Unix mode is on or off.

Returns

A value greater than zero when Unix is on and zero when Unix is off.

See Also

- unixpath
- unixcmd
- todospath
- todoscmd

tc "Function TODOSPATH"§xe "todospath"§**char *todospath(path)**

char *path /* Unix path to convert */

Convert a Unix path name to a DOS path name, when Unix mode is enabled.

Parameter	Description
path	Address of a NULL terminated character string containing a Unix path. This path can be relative or absolute.

Returns

The address of a NULL terminated string containing the DOS path name, when Unix mode is enabled, otherwise, the Unix path name is returned.

Comments

Path names are stored in a static buffer and is over-written each time this function is called.

When Unix mode is not enabled, then the Unix path name is simply copied into the static buffer and is not converted.

Typically, isunix() will if be called before todospath() to determine whether or not a path should be converted.

See Also

unixpath
unixcmd
isunix
todoscmd

tc "Function TODOSCMD"§xe "todoscmd"§**char *todoscmd(cmd)**

char *cmd /* Unix command to convert */

Convert a Unix command line to a DOS command line, when Unix mode is enabled.

Parameter	Description
-----------	-------------

cmd

Address of a NULL terminated character string containing a Unix command line.

Returns

The address of a NULL terminated string containing the DOS command line, when Unix mode is enabled, otherwise, the Unix command line is returned.

Comments

Command lines are stored in a static buffer and is over-written each time this function is called.

When Unix mode is not enabled, then the Unix command line is simply copied into the static buffer and is not converted.

Typically, `isunix()` will be called before `todoscmd()` to determine whether or not a command line should be converted.

See Also

[unixpath](#)
[unixcmd](#)
[isunix](#)
[todospath](#)

tc "File Description Functions" § **File Description Functions**

tc "Function GETDESC" §xe "getdesc" §**char *getdesc(path)**

char *path /* path of file */

Retrieve a file description for the specified file.

Parameter
path

Description

Address of a NULL terminated character string containing the full path of the described file.

Returns

The address of a NULL terminated string containing the file description. On error an empty string is returned.

Comments

The file description is stored in a static buffer and is over-written each time this function is called.

See Also

setdesc
deldesc

tc "Function SETDESC"§xe "setdesc"§**int setdesc(path, str)**

char *path /* path of file */
char *str /* character string */

Set a file description for the specified file.

Parameter	Description
path	Address of a NULL terminated character string containing the full path of the file to describe.
str	Address of a NULL terminated character string containing the file description. The description can contain a maximum of 60 characters.

Returns

On success a value of zero is returned. On error -1 is returned.

See Also

getdesc
deldesc

tc "Function DELDESC"§xe "deldesc"§int **deldesc(path)**

char *path /* path of file */

Delete a file description for the specified file.

Parameter	Description
path	Address of a NULL terminated character string containing the path of the described file.

Returns

A value of zero on success. On error -1 is returned.

See Also

getdesc
setdesc

tc "Environment Functions"§**Environment Functions**

tc "Function GETENVIRONMENT"§xe "getenv"§xe "getenvironment"§**char *getenvironment(name)**

char *name /* name of environment variable to retrieve */

Retrieve an environment variable from the WinOne environment space.

Parameter	Description
name	Address of a NULL terminated character string that contains the name of the environment variable to retrieve.

Returns

The address of a NULL terminated character string, where the environment variable is stored. On error a NULL value is returned.

Comments

The environment variable is stored in a static buffer, that is over-written each time this function is called.

See Also

putenvironment

tc "Function PUTENVIRONMENT"§xe "putenv"§xe "putenvironment"§**int**
putenvironment(name)

char *name /* environment string */

Place an environment string into the WinOne environment space.

Parameter	Description
name	Address of a NULL terminated character string that contains the environment string to place into the WinOne environment space.

Returns

Value of zero. On error, a value of -1 is returned

Comments

The parameter name is duplicated using malloc() by WinOne and therefore, does not need to be a static or malloced and not freed, which is the case with the standard library function putenv().

Environment strings have the form:-

VARNAME=ENVSTRING

To delete an environment variable, exclude the ENVSTRING. For example, to delete the environment variable AVAR :-

putenvironment("AVAR=");

See Also

getenvironment

tc "System Functions"§ **System Functions**

tc "Function SYSTEM"§xe "system"§**int system(command)**

char far *command /* command line to execute */

Execute a command or a program from within an external command.

Parameter	Description
command	Far address of a NULL terminated character string that contains the command line to execute. The command line can include anything that can be entered at the WinOne prompt, including executing another external command.

Returns

The errorlevel set be the executed command or program. It value depends upon the program been executed.

Index

µ_

_argc, 17

_argv, 17

A

argabs, 21

argc, 18

argn, 22

argnstr, 23

argpath, 20

args, 4, 22

argstr, 23

argtail, 22

argv, 19

atoc, 15

autogen, 4

C

clreol, 12

clrscr, 12

comma, 15

commas, 15

D

deldesc, 41

E

empty, 25

errcnt, 11

errorlevel, 4

errorlevels, 4

exit, 4

F

filecat, 27

filecpy, 26

filencat, 29

filencpy, 28

filesize, 25

fillfile, 30

fillpath, 34
fillpathall, 35
freepaths, 36

G

getch, 17
getchar, 16
getdesc, 40
getenv, 41
getenvironment, 41
getfile, 31
getfilename, 33
getfilepath, 33
getpath, 36
gets, 15
gotoxy, 13

I

inc, 25
isargstr, 24
isunix, 38

L

limit, 24

P

pad, 34
padfilename, 34
perror, 11
printf, 8
putch, 9
putchar, 10
putenv, 42
putenvironment, 42
puts, 9

S

scanf, 15
screen, 7
screenheight, 14
screenwidth, 14
scrheight, 14
scrwidth, 14
setdesc, 40

system, 42

T

textcolor, 11
todoscmd, 39
todopath, 38
totalfile, 36
totalfiles, 36

U

unixcmd, 37
unixpath, 37

W

wherex, 13
wherey, 14

Y

yield, 7

