WOIO Library©

Version 4.1

Introduction

General Information

Programming Information

External Commands

Function Reference

Function Reference Categories

Control Functions

screen	enable/disable buffered screen output
yield	Allow other tasks to multitask.

Screen Output Functions

printf	formatted output
puts	display a string
putchar	display a character
putch	display a character (direct video)
textcolor	change foreground colours
perror	display an error message
errcnt	global variable
clrscr	clear the screen
clreol	clear till the end of the line
gotoxy	set the caret screen position
wherex	get the horizontal caret position
wherey	get the vertical caret position
scroll	scroll a number of line up or down
insline	insert a line
delline	delete a line
scrflush	update the screen
scrwidth	get the screen width
scrheight	get the screen height
atoc	delimit a number with comma's

Input Functions

scanf	formatted input
gets	get a string
getchar	get a character
getch	get a character (direct keyboard)

Command Line Functions

argc	get the number of command line arguments
argv	retrieve a command line argument
arg_c	Global variable
arg_v	Global variable
argpath	retrieve a command line path
argabs	retrieve an absolute command line path
argtail	retrieve the actual command line tail
argn	get the number of command line switches
args	retrieve all the command line switches

Status Bar Functions

limit	set the status bar target value
inc	update the status bar by a value
empty	clear the status bar percent indicator

File Manipulation Functions

filesize	get the size of a file
istextfile	check for a text file
filecpy	copy a file
filencpy	copy part of a file
filecat	concatenate a file
filencat	concatenate part of a file

File Name Functions

fillfile	create a file table of all files in a given path
getfile	retrieve a file control block from the file table
getfilepath	retrieve a file path from the file table
getfilename	retrieve a file name from the file table
padfilename	pad a file name suitable for displaying

Path Name Functions

fillpath	create a path table of all directories
fillpathall	create a path table of all directories for all drives
freepaths	free the path table
getpath	retrieve a path from the path table

Unix Functions

todos	convert a Unix command to a DOS command
tounix	convert a DOS command to a Unix command
isunix	determine if Unix mode is on or off

File Description Functions

getdesc	get a file description
setdesc	set a file description
deldesc	delete a file description

Environment Functions

getenvironment	get a WinOne environment variable
putenvironment	set a WinOne environment variable

The WOIO package

WOIO is a library that allows programs to interact (ie. preform 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 then 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 known 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 accomplish with 5 lines of code :-

```
#include "woio.h"

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

    return 0;
}
```

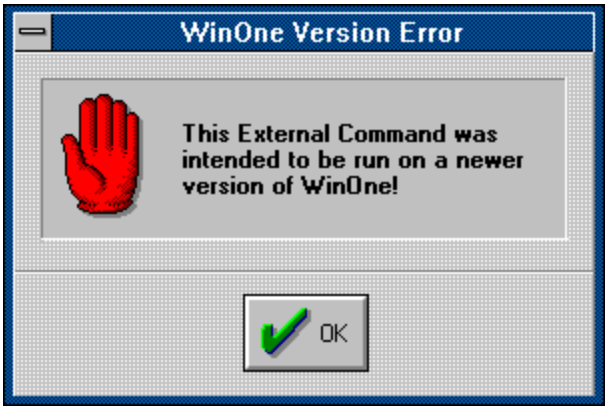
There are some additional things that need to be set-up correctly before the above program will work, for example, the project file, renaming the .EXE to a .EXC, etc. Apart from these extra things that are needed, some programs will only need to be modified to include this library to compile. However this will NOT generally be the case.

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.

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 :-



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 :-

argc	argn	argpath
argv	args	argabs
argtail		

The exit function exit() and Errorlevels

The standard library `exit()` may be used to exit a program at any time during it execution and the value past 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 then 0 (ie. excluding 3 and 255) to signal an error :-

```
#include "woio.h"

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

    return 0;       /* normal termination */
}
```

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	argpath	arg_c
argv	args	argabs	arg_v
argtail			

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.

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 :-

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

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).

Control Functions

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 */
}
```

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 */
}
```

Screen Output Functions

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()` :-

```
/*
   all external commands should use the following system
   colours instead of the fix colours
*/

#define COL_FILENAME    (char )144   /* system text colours */
#define COL_HIGHFNAME   (char )145
#define COL_NUMBER      (char )146
#define COL_TEXT        (char )147
#define COL_HIGHTEXT    (char )148
#define COL_BOLDTEXT    (char )149
#define COL_ENVNAME     (char )150
#define COL_ENVSTR      (char )151
#define COL_ERROR       (char )152
#define COL_LHS         (char )153
#define COL_HIGHLHS     (char )154
#define COL_RHS         (char )155
#define COL_FILEDATE    (char )156
#define COL_FILETIME    (char )157
#define COL_FILEATTRIB  (char )158
#define COL_FILEDESC    (char )159

#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?",
           COL_FILETIME, COL_FILEATTRIB);
```

will display the following :-

```
Hello, world
How are you?
```

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

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(COL_ERROR);       /* display message in COL_ERROR */

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

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

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');
}
```

int errcnt

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

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

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 :-

```
/*
   all external commands should use the following system
   colours instead of the fix colours
*/

#define COL_FILENAME    (char )144   /* system text colours */
#define COL_HIGHFNAME   (char )145
#define COL_NUMBER      (char )146
#define COL_TEXT        (char )147
#define COL_HIGHTEXT    (char )148
#define COL_BOLDTEXT    (char )149
#define COL_ENVNAME     (char )150
#define COL_ENVSTR      (char )151
#define COL_ERROR       (char )152
#define COL_LHS         (char )153
#define COL_HIGHLHS     (char )154
#define COL_RHS         (char )155
#define COL_FILEDATE    (char )156
#define COL_FILETIME    (char )157
#define COL_FILEATTRIB  (char )158
#define COL_FILEDESC    (char )159

#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

void clrscr(void)

Clear the screen.

Returns
There is no return value

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

void clreol(void)

Clear till the end of the current line.

Returns
There is no return value

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", COL_HIGHTEXT, s);

    gotoxy(0, scrheight());

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

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

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

void scroll(start, end, num)

int start /* position of line */
int end /* position of line */
int num /* number of places to scroll */

Scroll a range of line, a specified number of places, up or down on the screen.

Returns
There is no return value

Comments
The first line on the screen is at position 0.

A positive value for num, scrolls the range of lines downward and a negative value for num, scrolls the range of lines upward.

void insline(at)

int at /* position of line */

Insert a blank line at the specified position.

Returns
There is no return value

Comments
The first line on the screen is at position 0.

void delline(at)

int at /* position of line */

Delete a line at the specified position.

Returns
There is no return value

Comments
The first line on the screen is at position 0.

void scrflush(void)

Force updating of the screen.

Returns
There is no return value

int scrwidth(void)

Determine the screen width in characters.

Returns
The screen width in characters.

See Also
scrheight

int scrheight(void)

Determine the screen height in characters.

Returns
The screen height in characters.

See Also
scrwidth

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.

Input Functions

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

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 ", COL_HIGHTEXT, COL_TEXT);

    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;
}
```

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 get`char()` 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

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

Command Line Functions

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

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

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

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", COL_HIGHTEXT, COL_NUMBER, total);

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

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:*.*

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:\.

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

int argn(void)

Determines the number of command line switches.

Returns
The number of switches.

See Also
argc
argv
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

Status Bar Functions

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

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

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

File Manipulation Functions

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

int istextfile(path)

char *path /* file name path */

Determine whether the specified file contains text.

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
TRUE when the specified file is a text file, otherwise FALSE is returned. On error -1 is returned.

Comments
On error `istextfile()` does NOT display an error message.

Function `istextfile()` check's the first 100 bytes of the specified file to determine if it is a text file or not.

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:-

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 `filecpy()` will fail.

These values can NOT be combined.

u_flag	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, `filecpy()` 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
filencpy
filecat
filencat

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	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 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

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:-

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.

These values can NOT be combined.

u_flag	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

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:-

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 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

File Name Functions

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

BOOL getfile(index, pff)

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

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

Parameter	Description
index	Specifies which ffblk 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 ffblk {
        char ff_reserved[21];   /* reserved by DOS */
        char ff_attrib;         /* attribute found */
        int   ff_ftime;                 /* 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 ffblk ffblk;

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

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

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

        if (getfile(i, &ffblk) == FALSE) {
            perror("Bad index");
```

```
            return 1;
        }

        printf(" %c%-13s%c%9ld\n",
            COL_FILENAME, ffblk.ff_name,
            COL_NUMBER, ffblk.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 */
}
```

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

char *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

char *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.

Path Name Functions

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

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

void freepaths(void)

Release the memory allocated to store the Path Table.

Returns
There is no return value.

See Also
- fillpath
- fillpathall

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

Unix Functions

char *tounix(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.

This function is also used to convert DOS paths to Unix paths, suitable for displaying.

See Also
isunix
todos

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
tounix
todos

char *todos(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
This function overwrites the cmd parameter.

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

This function is also used to convert Unix paths to DOS paths.

See Also
tounix
isunix

File Description Functions

char *getdesc(path)

char *path /* path of file */

Retrieve a file description for the specified file.

Parameter	Description
path	Address of a NULL terminated character string containing the full path of the decribed 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

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

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

Environment Functions

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

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