

## Library Listings

### Alphabetic Function Listing

arm_guard()	11-9	mask_and()	11-64	SVC_INFO_BAR()	11-123
bad_ptr()	11-11	mask_clear()	11-65	SVC_INFO_CARD()	11-125
_clean_list()	11-12	mask_empty()	11-66	SVC_INFO_DSP()	11-127
close()	11-14	mask_fill()	11-67	SVC_INFO_EPPROM()	11-129
close_all()	11-16	mask_in()	11-68	SVC_INFO_KEY()	11-130
cfileol()	11-17	mask_of()	11-69	SVC_INFO_PLATFORM()	11-132
cifscr()	11-18	mask_or()	11-70	SVC_INFO_PTID()	11-133
creat()	11-19	open()	11-71	SVC_INFO_TYPE()	11-134
delete()	11-21	open_codefile()	11-73	SVC_INT2()	11-136
delete_cvir()	11-23	pending_traps()	11-75	SVC_KEY_NUM()	11-137
delete_vir()	11-25	put_env()	11-76	SVC_KEY_TXT()	11-139
delline()	11-27	putkey()	11-77	SVC_MOD_CK()	11-141
dir_get_first()	11-28	put_lan_config()	11-79	SVC_PACK4()	11-143
dir_get_next()	11-29	putpixelcol()	11-81	SVC_RAM_SIZE()	11-145
dir_get_sizes()	11-30	read()	11-83	SVC_RESTART()	11-146
disarm_guard()	11-31	read_cmd()	11-85	SVC_STORE()	11-149
dispatch_guard()	11-33	read_cvir()	11-87	SVC_TICKS()	11-151
_free()	11-35	read_reject_pkt()	11-89	SVC_2INT()	11-153
file_copy()	11-37	read_vir()	11-91	SVC_UNPK4()	11-154
getcontrast()	11-39	resetdisplay()	11-93	SVC_VERSION()	11-156
get_env()	11-40	seek_cvir()	11-94	SVC_WAIT()	11-157
getfont()	11-41	seek_vir()	11-96	SVC_ZONTALK()	11-158
getfontinfo()	11-42	setcontrast()	11-98	SVC_ZONTALK() LAN...	11-160
getgrid()	11-44	set_env_buffer()	11-100	tick_compare()	11-162
getkey()	11-45	setfont()	11-102	trap_of()	11-163
get_lan_config()	11-47	setscrollmode()	11-104	unlink()	11-164
getscrollmode()	11-50	sound()	11-106	wherecur()	11-165
gotoxy()	11-52	SVC_CHECKFILE()	11-108	wherewin()	11-166
insert()	11-54	SVC_CHK_PASSWORD()	11-110	wherewincur()	11-168
insert_cvir()	11-56	SVC_CLOCK()	11-111	which_guard()	11-169
insert_vir()	11-58	SVC_COUNT()	11-113	window()	11-171
insline()	11-60	SVC_CRC_CALC()	11-117	write()	11-173
ioctl()	11-61	SVC_DSP_2_HEX()	11-119	write_cmd()	11-175
ioctlc()	11-30	SVC_GET_UNIT_ID()	11-121	write_cvir()	11-177
lseek()	11-62	SVC_HEX_2_DSP()	11-122	write_vir()	11-179

## Functions by Category

<b>Guard, Mask and Trap Functions</b>	<b>Page Number</b>
<code>arm_guard()</code>	Calls function when trappable event occurs ..... 11-9
<code>disarm_guard()</code>	Reverse effect of <code>arm_guard()</code> ..... 11-31
<code>dispatch_guard()</code>	Calls function defined in <code>arm_guard()</code> ..... 11-33
<code>mask_and()</code>	Logical AND operation on two trap masks ..... 11-64
<code>mask_clear()</code>	Resets the specified bits in a trap mask ..... 11-65
<code>mask_empty()</code>	Returns trap mask with all bits set to 0 ..... 11-66
<code>mask_fill()</code>	Returns trap mask with all bits set to 1 ..... 11-67
<code>mask_in()</code>	Performs a bit test on a trap mask ..... 11-68
<code>mask_of()</code>	Sets a bit in a trap mask ..... 11-69
<code>mask_or()</code>	Logical OR operation on two trap masks ..... 11-70
<code>pending_traps()</code>	Reads and/or updates traps mask ..... 11-75
<code>trap_of()</code>	Returns the trap number ..... 11-163
<code>which_guard()</code>	Returns the address of a guard function ..... 11-169

<b>General Display Functions</b>	<b>Page Number</b>
<code>clearol()</code>	Clears display line from cursor position ..... 11-17
<code>clearscr()</code>	Clears the current display window ..... 11-18
<code>delline()</code>	Deletes the display line containing cursor ..... 11-27
<code>getscrollmode()</code>	Gets display scrolling settings ..... 11-50
<code>gotoxy()</code>	Moves the cursor to a specified position ..... 11-52
<code>insline()</code>	Inserts a blank line ..... 11-60
<code>setscrollmode()</code>	Sets the display scrolling mode ..... 11-104
<code>wheretur()</code>	Gets cursor position ..... 11-165
<code>wherewin()</code>	Gets display window coordinates ..... 11-166
<code>wherewincur()</code>	Gets cursor position relative to window ..... 11-168
<code>window()</code>	Defines a window ..... 11-171
<code>write()</code>	Transfers data to file, or device buffer* ..... 11-173

\* Function listed in more than one category.

<b>Pixel-type Display Functions</b>	<b>Page Number</b>
<code>getcontrast()</code>	Gets contrast level ..... 11-39
<code>getfont()</code>	Gets the name of the Font Definition File ..... 11-41
<code>getfontinfo()</code>	Gets font ROM information ..... 11-42
<code>getgrid()</code>	Gets current grid setting ..... 11-44
<code>putpixelcol()</code>	Displays graphic images byte-by-byte ..... 11-81
<code>resetdisplay()</code>	Sets the font and grid ..... 11-93
<code>setcontrast()</code>	Sets contrast level ..... 11-98
<code>setfont()</code>	Changes current font ..... 11-102

<b>Device Functions</b>	<b>Page Number</b>
<code>close()</code>	Terminates usage of file or device handle ..... 11-14
<code>get_lan_config()</code>	Retrieves LAN parameters ..... 11-47
<code>ioctl()</code>	Perform file or device-specific functions ..... 11-61
<code>ioctlc()</code>	Perform file or device-specific functions ..... 11-30
<code>open()</code>	Prepares a file or device for processing* ..... 11-71
<code>put_lan_config()</code>	Configures LAN parameters ..... 11-79
<code>read()</code>	Transfers file or device data to buffer ..... 11-83
<code>read_cmd()</code>	Reads a modem command response ..... 11-85
<code>read_reject_pkt()</code>	Retrieve the next rejected transmit attempt ..... 11-89
<code>sound()</code>	Generates various notes ..... 11-106
<code>SVC_CRC_CALC()</code>	Calculates a Cyclic Redundancy Check (CRC)* .... 11-117
<code>SVC_INFO_BAR()</code>	Gets bar code reader information* ..... 11-123
<code>SVC_INFO_CARD()</code>	Gets card reader information* ..... 11-125
<code>SVC_INFO_DSP()</code>	Gets information about the display device* ..... 11-127
<code>SVC_INFO_KEY()</code>	Gets information about the keyboard* ..... 11-130
<code>SVC_KEY_NUM()</code>	Allow entry of a formatted decimal number* ..... 11-137
<code>SVC_KEY_TXT()</code>	Allow entry of formatted data* ..... 11-139
<code>SVC_MOD_CHK()</code>	Validate or generate the Luhn check* ..... 11-141

\* Function listed in more than one category.

SVC_ZONTALK()	Executes a ZONTALK 2000 download* .....	11-158
write()	Transfers data to file, or device buffer* .....	11-173
write_cmd()	Sends modem commands .....	11-175

**File and Directory Functions** *Page Number*

close()	Terminates usage of a file or device handle .....	11-14
close_all()	Release all active file handles .....	11-16
creat()	Create new file or rewrite existing .....	11-19
delete()	Deletes specified data from a file .....	11-21
delete_cvir()	Deletes compressed variable-length records .....	11-23
delete_vir()	Deletes variable-length records .....	11-25
dir_get_first()	Gets name of the first file in directory .....	11-28
dir_get_next()	Gets the name of the next file in directory .....	11-29
dir_get_sizes()	Gets number of files and bytes .....	11-30
file_copy()	Copies a source file to a destination file .....	11-37
get_env()	Gets the value of an environment variable .....	11-40
getkey()	Retrieves the data keyed file .....	11-45
insert()	Inserts data from a buffer into a file .....	11-54
insert_cvir()	Inserts compressed variable-length record .....	11-56
insert_vir()	Inserts variable-length record into a file .....	11-58
ioctl()	Perform file or device specific functions* .....	11-61
lseek()	Moves the file position pointer .....	11-62
open()	Prepares a file or device for processing* .....	11-71
open_codefile()	Opens a file as an executable code file .....	11-73
put_env()	Stores an environment variable .....	11-76
putkey()	Stores the data into a keyed file .....	11-77
read()	Transfers file or device data to buffer .....	11-83
read_cvir()	Reads a compressed variable-length record .....	11-87
read_vir()	Reads a variable-length record .....	11-91
seek_cvir()	Sets the compressed file position pointer .....	11-94

\* Function listed in more than one category.

<code>seek_vlr()</code>	Sets the file position pointer .....	11-96
<code>set_env_buffer()</code>	Sets user environment variable buffer.....	11-100
<code>unlink()</code>	Delete file.....	11-164
<code>write()</code>	Transfers data to file, or device buffer* .....	11-173
<code>write_cvlr()</code>	Writes compressed variable-length record .....	11-177
<code>write_vlr()</code>	Writes variable-length record.....	11-179

**System Services Functions**

	<i>Page Number</i>	
<code>SVC_CHECKFILE()</code>	Check file checksum .....	11-108
<code>SVC_CHK_PASSWORD()</code>	Compares string to system password.....	11-110
<code>SVC_CLOCK()</code>	Read or change the current time.....	11-111
<code>SVC_COUNT()</code>	Access diagnostics statistical counts.....	11-113
<code>SVC_CRC_CALC()</code>	Calculates Cyclic Redundancy Check (CRC)* .....	11-117
<code>SVC_DSP_2_HEX()</code>	Converts ASCII-hex to hexadecimal.....	11-119
<code>SVC_GET_UNIT_ID()</code>	Gets terminal unit identification number .....	11-121
<code>SVC_HEX_2_DSP()</code>	Converts hexadecimal to ASCII-hex.....	11-122
<code>SVC_INFO_BAR()</code>	Gets bar code reader information* .....	11-123
<code>SVC_INFO_CARD()</code>	Gets card reader information* .....	11-125
<code>SVC_INFO_DSP()</code>	Gets information about the display device* .....	11-127
<code>SVC_INFO_EPROM()</code>	Gets EPROM (firmware) version number.....	11-129
<code>SVC_INFO_KEY()</code>	Gets information about the keyboard* .....	11-130
<code>SVC_INFO_PLATFORM()</code>	Gets information about the system type .....	11-132
<code>SVC_INFO_PTID()</code>	Gets the permanent terminal identifier .....	11-133
<code>SVC_INFO_TYPE()</code>	Gets processor and terminal information.....	11-134
<code>SVC_INT2()</code>	Converts a binary number to ASCII-decimal .....	11-136
<code>SVC_KEY_NUM()</code>	Allow entry of a formatted decimal number* .....	11-137
<code>SVC_KEY_TXT()</code>	Allow entry of formatted data* .....	11-139

\* Function listed in more than one category.

## Library Functions

SVC_MOD_CHK()	Validate or generate the Luhn check* .....	11-141
SVC_PACK4()	Compresses buffers writing to file .....	11-143
SVC_RAM_SIZE()	Returns amount of static RAM .....	11-145
SVC_RESTART()	Starts executing a new code file .....	11-146
SVC_STORE()	Change or create a keyed file .....	11-149
SVC_TICKS()	Read the system's tick counter .....	11-151
SVC_2INT()	Converts ASCII decimal to binary .....	11-153
SVC_UNPK4()	Decompresses the output of SVC_PACK40 .....	11-154
SVC_VERSION()	Gets the terminal's EPROM ID .....	11-156
SVC_WAIT()	Causes a software delay .....	11-157
SVC_ZONTALK()	Initiates a ZONTALK 2000 download* .....	11-158
tick_compare()	Compares two timer values .....	11-162

## Miscellaneous Functions

	Page Number	
bad_ptr()	Checks pointer returned from malloc() .....	11-11
_clean_list()	Cleans up the free list and cuts back the heap pointer .....	11-12
_free()	Deallocates memory allocated by malloc() .....	11-35

See *Standard C Programming Language for TXO Reference Manual*, VeriFone part number 11469 for: free(), getenv(), remove().

\* Function listed in more than one category.

## Standard C for TXO Library Listing

See Standard C Programming Language for TXO Reference Manual, Part No. 11469

abs()	fread()	longjmp()	sscanf()
asin()	free()	lseek()	strcat()
atan()	freopen()	malloc()	strchr()
atan2()	fscanf()	memccpy()	strcmp()
atof()	fseek()	memchr()	strcpy()
atoi()	ftell()	memcmp()	strcspn()
atol()	fwrite()	memcpy()	strlen()
calloc()	getc()	memset()	strncat()
ceil()	getchar()	modf()	strncmp()
clearerr()	getenv()	open()	strncpy()
close()	gets()	pow()	strpbrk()
cos()	getw()	printf()	strrchr()
cosh()	hypot()	putc()	strspn()
creat()	insert()	putchar()	strstr()
delete()	ioctl()	puts()	strtod()
exp()	isalnum()	putw()	strtok()
fabs()	isalpha()	qsort()	swab()
fclose()	isascii()	rand()	tan()
fdopen()	isctrl()	read()	tanh()
feof()	isdigit()	remove()	toascii()
ferror()	islower()	rename()	tolower()
fflush()	isprint()	rewind()	toupper()
fgetc()	ispunct()	scanf()	ungetc()
fgets()	isspace()	setjmp()	unlink()
floor()	isupper()	sin()	va_arg()
fopen()	isxdigit()	sinh()	va_end()
fprintf()	ldexp()	sprintf()	va_start()
fputc()	log()	sqrt()	write()
fputs()	log10()	strand()	

## arm\_guard()

Calls a function pointed to by `proc` when a trappable event specified in `take` occurs, passing the guard slot number on success to the `proc` function.

Each `arm_guard()` issued by the application is inserted into a list of guards, with the most recently issued `arm_guard()` at the top of the list.

**Prototype**

```
#include <trap.h>
int arm_guard( proc, take, hide );
void *proc;
struct trap_mask take, hide;
```

**Parameters**

`proc` is a pointer to the application's guard function, which takes the trap number associated with the trappable event defined in the `take` parameter.

`take` is a `trap_mask` structure (defined in `<trap.h>`) identifying the events this guard will monitor. When the event occurs, its associated guard function is called to process the event.

`hide` is a `trap_mask` structure identifying the events to be ignored while the guard function is processing.

**Returns**

<code>slot_number</code>	Guard slot number if registration is successful.
<code>-1</code>	Failure.

The guard slot number may be used as the parameter to the `disarm_guard()` function.

**Notes**

When a trappable event occurs, the operating system searches the guard list from top to bottom for the trap number specified by the `take` parameter. The first specified guard encountered triggers its associated function to process the event, and the trap is reset. Thus, if multiple guards are set up to monitor the same trap, only the most recently armed guard is handled and reset.



If another trappable event occurs while the current guard function is executing, the hide mask is checked to see if the new event should be ignored. Event(s) listed in the hide parameter are of lower priority than the event(s) listed by the take parameter.

Typically hide and take share the same value so that a recurring event is not handled twice.

To block interruptions while a guard function executes, use the hide mask produced by the mask\_fill() routine, page 11-67.

**Related** disarm\_guard(), dispatch\_guard(), which\_guard()

**Example**

```
/* ARMGUARD.C */
#include <stdio.h>
#include <trap.h>
#include <txosvc.h>

/* Routine called when clear key is pressed. */
void clear_key_trap(trap_number)
int trap_number;
{
    printf("\fRECEIVED TRAP %d", trap_number);
    SVC_WAIT(1000);
    printf("\fCLEAR PRESSED");
    SVC_WAIT(1000);
    printf("\fPRESS CLEAR KEY");
}
struct trap_mask take;
int slot;
main () {
    /* Define take mask for trap 32. */
    take=mask_of(32);
    /* Arm guard for clear key presses. */
    slot=arm_guard(clear_key_trap,take,take);
    /* Loop forever, calling trap routine when clear pressed. */
    printf("\f%d: PRESS CLEAR", slot);
    while (1);
}
```

## bad\_ptr()

Checks the validity of a pointer returned from `malloc()`.

**Prototype** `#include <unistd.h>`

```
int bad_ptr( pointer );
char *pointer;
```

**Parameters** `pointer` is the pointer to be tested.

**Returns** `TRUE` `pointer` is null, appears to not have been allocated via `malloc()`, or appears to have been freed already.

`FALSE` (zero) Suggests that this is a valid pointer to an allocated item which can be freed.

**Notes** It is not always possible to detect bad pointers.

### Example

```

/* BAD_PTR.C */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

char *p, *malloc();

main () {
    /* Allocate 20 bytes of memory. */
    p = malloc(20);

    /* Pointer should be good because it
       just came from malloc. */
    if (!bad_ptr(p)) printf("\fGood Pointer");
    getchar();

    /* Free pointer, then check for validity;
       it should be bad now. */
    free(p);
    if (bad_ptr(p)) printf("\fBad Pointer");
}

```

## \_clean\_list()

Cleans up the free list and cuts back the heap pointer as far as possible.

**Prototype** <txostd.h>  
 int \_clean\_list(void);

**Parameters** None

**Returns** TRUE Free list appears to have been cleaned up and unneeded space released successfully.  
 (non-zero)  
 FALSE Free list appears to be corrupted.  
 (zero)

**Notes** The \_clean\_list() routine attempts to clean up the free list by merging all adjacent free blocks and returning all contiguous free space from the heap to the stack. If all allocated items have been freed, the heap pointer can be cut back as far as the Heap Base + 2.

There may be forms of heap corruption which are not detectable and are reported as TRUE even though the list is corrupt. If the links within the heap have been corrupted due to the program writing before the beginning or after the end of allocated data areas, the heap is corrupted. In this case, \_clean\_list() may not be able to detect it.

If you do not reference \_clean\_list() and use free(), space allocated to the heap is kept on the heap, not returned to the stack. In this case, both free() and malloc() calls run faster, and the \_clean\_list() routine is unnecessary. In terms of speed and efficiency, the best compromise may be to call the \_clean\_list() routine periodically, or after you free() a large item or a large number of items.

To automatically return all possible space to the stack every time you free a memory allocation, you may declare your own free() routine, as shown in the example on the following page.

**Example**

```
/* CLEANLST.C */
#include <stdlib.h>
#include <txostd.h>

main()
{
    char *p;

    p=(char *)malloc(10);

    /* If freeing pointer succeeds */
    if (!_free(p))
    /* return space to stack */
    (void)_clean_list();
}
```

## close()

---

Terminates use of a file or device handle.

**Prototype** #include <io.h>  
int close( handle );  
int handle;

**Parameters** handle is the value returned by open().

**Returns** 0 Success.

-1 Failure, with errno set to a specific error value.

**Notes** If device writes via write() are pending or in progress while close() is called, data may be lost.

**Example**

```
/* FILE.C */
#include <stdio.h>
#include <io.h>
#include <fcntl.h>
int file_handle;
char buffer[20];
main () {
    /* Create file for read and write. */
    file_handle = open("Test.Dat", 0_CREAT+0_RDWR);

    /* Write to the file. */
    write(file_handle, "Hello World", 11);

    /* Close the file */
    close(file_handle);

    /* Open the file for read only. */
    file_handle = open("Test.Dat", 0_RDONLY);

    /* Read from the file. */
    read(file_handle, buffer, 11);

    /* Display the contents of the buffer. */
    buffer[11]=0;
    printf("\f%s",buffer);

    /* Close the file */
    close(file_handle);
}
```

## close\_all()

Releases all active file handles currently in use by the file system. Devices are unaffected by this function.

**Prototype** #include <txostd.h>

```
int close_all(void);
```

**Parameters** None

**Returns** 0 Success, returned in all cases.

**Notes** Buffered file I/O data, for example, pending data written by fwrite(), will be lost following a call to close\_all().

### Example

```
/* CLOSEALL.C */
#include <fcntl.h>
#include <txostd.h>

int i, file_handle[10];
char filename[2];
main () {
    /* Filenames will be 'A', 'B', 'C', etc. */
    filename[0] = 'A';
    filename[1] = 0;

    /* Create 10 files. */
    for (i=0; i<10; i++) {
        file_handle[i] = open(filename, O_CREAT+O_RDWR);
        write(file_handle[i], filename, 1);
        filename[0] = filename[0]+1;
    }

    /* Close all 10 files. */
    close_all();
}
```

## clreol()

Clears the display line in the current window from the cursor position to the end of the line.

**Prototype** #include <txostd.h>  
void clreol(void);

**Parameters** None

**Returns** VOID

**Notes** Cursor position is unchanged.

**Pixel-type display:** This function clears to the end of line by writing spaces in the current font. If the current font is in reverse video, the line will be cleared by writing reverse video space, and thus appears darkened.

**Related** window(), gotoxy()

### Example

```
/* CLEARING.C */
#include <stdio.h>
#include <txostd.h>

main () {
    /* Fill the display. */
    printf("\f1234567890ABCDEF");
    /* Move to the 'A' and wait. */
    gotoxy(11,1);
    getchar();
    /* Clear to the end of line and wait. */
    clreol();
    getchar();
    /* Clear the screen. */
    clrscr();
}
```

clreol()

11-17



## clrscr()

---

Clears the current display window and places the cursor in the window's upper left hand corner (column 1, line 1).

**Prototype** #include <txostd.h>

void clrscr(void);

**Parameters** None

**Returns** VOID

**Notes** Pixel-type display: This function clears the window by writing space characters from the current font. If the current font is in reverse video, the space character may be a solid space (darkened) instead of an empty space, causing this function to fill or darken the entire window.

**Related** gotoxy(), window()

**Example** See clr\_eol() example on the previous page.

## creat()

---

Generates either a new file or prepares to rewrite an existing file specified by *pathname*. If the file does not exist, a new file is created and opened. If the file already exists, `creat()` truncates the file to a length of zero bytes (flushes the file) and then opens the file for writing.

**Prototype**  
`#include <fcntl.h>`  
`#include <errno.h>`  
`int creat(pathname, permis);`  
`char *pathname;`  
`int permis;`

**Parameters**  
`pathname` Path name of new file.  
`permis` Permission setting — see Notes.

**Returns** A handle for the new file if the call is successful, or -1 if an error occurs. On error, `errno` is set to one of the following:

- EMFILE Too many open files.
- ENOENT Pathname not found.

**Notes** `creat()` looks at `permis` for permission to write to the file. In TXO C, however, the permission field is ignored, and may include any value.

**Related** `open()`

**Example** See following page.

Example

```
/* CREAT.C */  
#include <stdio.h>  
#include <fcntl.h>  
#include <errno.h>  
int file_handle;  
int bytes_written;  
extern int errno;  
  
main()  
{  
    /* Create a file */  
    file_handle = creat("Test.Dat", 0_RDWR);  
    bytes_written = write(file_handle, "hello world", 11);  
    close(file_handle);  
  
    /* Make sure it is here */  
    if ((file_handle = open("Test.Dat", 0_RDONLY)) != -1)  
        printf("FILE FOUND");  
    close(file_handle);  
}
```

## delete()

---

Deletes specified data from a file. Any data following is moved to fill the gap.

**Prototype**  
`#include <io.h>`  
`int delete( handle, count );`  
`int handle, count;`

**Parameters**  
`handle` is the value returned by `open()`.  
`count` is the number of bytes to delete, starting at the current file pointer position.

**Returns**  
-1 Error occurred during deletion, with `errno` containing a specific error value. Values for `errno` include:  
EBADF  
ENOENT  
EBUSY

No other meaningful value is returned.

**Notes** The file position pointer is not changed.

**Example** See following page.

Example

```
/* DELETE.C */
#include <stdio.h>
#include <io.h>
#include <fcntl.h>
int file_handle;
char buffer[20];
main () {
    /* Create file with read and write. */
    file_handle = open("Test.Dat", 0_CREAT+0_RDWR);
    /* Write to the file. */
    write(file_handle, "Hello Big World", 15);
    /* Seek to the beginning of the word "Big" */
    lseek(file_handle, 6, SEEK_SET);
    /* Delete "Big" to make "Hello World" */
    delete(file_handle, 4);
    /* Read from the file and display the result. */
    lseek(file_handle, 0, SEEK_SET);
    read(file_handle, buffer, 11);
    buffer[11] = 0;
    printf("\f%s",buffer);
    /* Close the file */
    close(file_handle);
}
```

## delete\_cvlr()

---

Deletes one or more compressed variable-length records from a file.

**Prototype**  
`#include <v1r.h>`  
`int delete_cvlr( handle, count );`  
`int handle, count;`

**Parameters**  
`handle` is the file handle returned by `open()`.  
`count` is the number of records to delete, starting at the current file position pointer.

**Returns**  
0 Success.  
-1 Failure, with `errno` set to a specific error value. Values for `errno` include:  
EBADF  
EINVAL

**Notes**  
Although files normally contain only one type of data, they may contain a mixture of binary data, normal variable-length records, and compressed variable-length records. For this function to delete the correct quantity of data, it is important that the file position indicator is positioned at the beginning of a compressed variable-length record.

The file position pointer is not changed.

**Example** See following page.

## Example

```

/* CVLR.C */
#include <stdio.h>
#include <io.h>
#include <fcntl.h>
#include <vwr.h>
int file_handle;
char buffer[20];
main () {
    /* Create file with read and write. */
    file_handle = open("Test.Dat", 0_CREAT+0_RDWR);

    /* Write compressed variable length records to the file.
*/
    write_cvlr(file_handle, "Hello ", 6);
    write_cvlr(file_handle, "Small ", 6);
    write_cvlr(file_handle, "World", 5);

    /* Seek to the beginning of the word "Small" */
    seek_cvlr(file_handle, 1, SEEK_SET);

    /* Delete "Small" */
    delete_cvlr(file_handle, 1);

    /* Insert "Big" */
    insert_cvlr(file_handle, "Big ", 4);

    /* Read from the file and display the result. */
    seek_cvlr(file_handle, 0, SEEK_SET);
    read_cvlr(file_handle, buffer, 6);
    read_cvlr(file_handle, buffer+6, 4);
    read_cvlr(file_handle, buffer+10, 5);
    buffer[15]=0;
    printf("\f%s",buffer);

    /* Close the file */
    close(file_handle);
}

```

## delete\_vlr()

---

Deletes one or more variable-length records from a file.

**Prototype** #include <vlr.h>  
int delete\_vlr( handle, count );  
int handle, count;

**Parameters** handle is the file handle returned by open().  
count is the number of records to delete, starting at the current file position pointer.

**Returns** 0 Success.  
-1 Failure, with errno set to a specific error value. Values for errno include:  
EBADF  
EINVAL

**Notes** Although files normally contain only one type of data, they may contain a mixture of binary data, variable-length records, and compressed variable-length records. For this function to delete the correct quantity of data, the file position indicator must be positioned at the beginning of a variable-length record.  
The file position pointer is not changed.

**Example** See following page.



## Example

```
/* VLR.C */
#include <stdio.h>
#include <io.h>
#include <fcntl.h>
#include <vlr.h>
int file_handle;
char buffer[20];
main () {
    /* Create file with read and write. */
    file_handle = open("Test.Dat", 0_CREAT+0_RDWR);

    /* Write variable length records to the file. */
    write_vlr(file_handle, "Hello ", 6);
    write_vlr(file_handle, "Small ", 6);
    write_vlr(file_handle, "World", 5);

    /* Seek to the beginning of the word "Small" */
    seek_vlr(file_handle, 1, SEEK_SET);

    /* Delete "Small" */
    delete_vlr(file_handle, 1);

    /* Insert "Big" */
    insert_vlr(file_handle, "Big ", 4);

    /* Read from the file and display the result. */
    seek_vlr(file_handle, 0, SEEK_SET);
    read_vlr(file_handle, buffer, 6);
    read_vlr(file_handle, buffer+6, 4);
    read_vlr(file_handle, buffer+10, 5);
    buffer[15]=0;
    printf("\f%s", buffer);

    /* Close the file */
    close(file_handle);
}
```

## delline()

---

Deletes the display line containing the cursor and moves all lines below it up one line.

**Prototype** #include <txostd.h>

void delline(void);

**Parameters** None

**Returns** VOID

**Notes** The cursor position is not changed.

### Example

```
/* DELLINE.C */
#include <stdio.h>
#include <txostd.h>

main () {
    /* Fill the display and wait. */
    printf("\f1234567890ABCDEDEF");
    getchar();
    /* Clear the screen using delline. */
    delline();
}
```

## dir\_get\_first()

Returns a null-terminated string containing the name of the first file found in the directory.

**Prototype** #include <txostd.h>  
 int dir\_get\_first( buffer );  
 char \*buffer;

**Parameters** buffer receives the name of the file.

**Returns** 0 Success.  
 Any other value should be treated as a severe error.  
 Values for errno include:  
 EFAULT

**Notes** buffer should be large enough to hold the largest possible file name and the terminating null byte (33 bytes).

**Related** dir\_get\_next()

### Example

```

/* DIR_GET.C */
#include <stdio.h>
#include <txostd.h>

char file[33];
main() {
    /* Display first file in
    the directory and wait. */
    dir_get_first(file);
    printf("\fFIRST=%s", file);
    getchar();

    /* Display next file in the directory. */
    dir_get_next(file);
    printf("\fNEXT=%s", file);
}
    
```

## dir\_get\_next()

Searches the directory for a specified file name, and returns the name of the next file in the directory.

**Prototype** #include <txostd.h>  
int dir\_get\_next( buffer );  
char \*buffer;

**Parameters** buffer receives the file name.

**Returns** An error value is returned if the specified file name is not found or it is the last file in the directory. Values for errno include:  
ENOENT

**Notes** This function is normally used when creating or displaying a list of all files in the directory. Typically, dir\_get\_first() is called once to obtain the name of the first file in the directory, followed by repeated calls to dir\_get\_next() to retrieve the remaining file names.

The destination buffer should be large enough to hold the largest possible file name and the terminating null byte (33 bytes).

**Related** dir\_get\_first()

**Example** See dir\_get\_first() example on the previous page.

## dir\_get\_sizes()

Returns the number of files in the directory, the number of bytes in use and the number of bytes remaining in the file system.

**Prototype** #include <txostd.h>  
void dir\_get\_sizes( buffer );  
char \*buffer;

**Parameters** buffer receives a structure containing the directory information.

**Returns** Ten bytes are copied into buffer:

int filecount;    Number of files in directory  
long currentsize;    Number of bytes in use  
long availablesize;    Number of bytes remaining

### Example

```

/* DIR_SIZE.C */
#include <stdio.h>
#include <txostd.h>

/* Buffer structure to receive directory information. */
union {
    struct s_dir {
        int filecount;    /* number of files in directory */
        long currentsize;    /* number of bytes in use */
        long availablesize;    /* number of bytes remaining */
    } directory;
    char dir_line[10];
} dline;

main() {
    /* Get directory information. */
    dir_get_sizes(dline.dir_line);

    /* Display number of files and wait for key press. */
    printf("\fFILES=%d",dline.directory.filecount);    getchar();

    /* Display number of bytes used and wait for key press. */
    printf("\fUSED=%ld",dline.directory.currentsize);    getchar();

    /* Display number of free bytes remaining. */
    printf("\fFREE=%ld",dline.directory.availablesize);
}

```

## disarm\_guard()

---

Reverses the effect of an `arm_guard()` call.

**Prototype** `#include <trap.h>`  
`int disarm_guard( slot );`  
`int slot;`

**Parameters** `slot` is the slot number returned by `arm_guard()` that `disarm_guard()` will inactivate.

**Returns** `0` `arm_guard()` no longer in effect.  
`-1` Invalid operation.

**Notes** Once armed, a guard remains active until it is disarmed. It is the application's responsibility to disarm all armed guards prior to program termination. If a program terminates in an abnormal manner, the operating system will disarm all guards.

**Related** `arm_guard()`, `dispatch_guard()`

**Example** See following page.

Example

```
/* DISARMGD.C */
#include <stdio.h>
#include <trap.h>
struct trap_mask take;
int slot, loop_flag;

/* Routine called when clear key is pressed. */
void clear_key_trap(trap_number)
int trap_number;
{
    loop_flag=0;
}

main () {
    /* Define take mask for trap 32. */
    take=mask_of(32);

    /* Arm guard for clear key presses. */
    slot=arm_guard(clear_key_trap,take,take);

    /* Loop until trap is executed. */
    printf("\fPRESS CLEAR KEY");
    loop_flag=1;
    while (loop_flag);

    /* Disarm guard. */
    if (disarm_guard(slot))
        printf("\fDISARM FAILED");
    else
        printf("\fDISARM OK");
}
```

## dispatch\_guard()

Explicitly calls a function defined in `arm_guard()`, whether or not the trappable event has occurred.

**Prototype** `#include <trap.h>`

```
void dispatch_guard( set_mask );
struct trap_mask set_mask;
```

**Parameters** `set_mask` is the trap\_mask structure set by `arm_guard()`, which is ORed with the system's pending trap mask when `dispatch_guard()` is called.

**Returns** VOID

**Related** `arm_guard()`, `disarm_guard()`

**Notes** This function calls one function for each `arm_guard()` with `dispatch_guard()`'s `set_mask` value matching `arm_guard()`'s take parameter.

This function sets user-defined traps (Traps 28-31).

### Example

```
/* DISPATCH.C */
/* This example arms a trap handler for two events:
/* CLEAR key and 1 second timer. It just sits in a
/* non-terminating loop incrementing a counter. When
/* the CLEAR key is pressed the trap handler will
/* display the value of the counter (long) and the
/* number of elapsed seconds. Just so things don't
/* overflow, when the counter gets to 50,000 we cause
/* the CLEAR key handler to run using dispatch_guard(). */
#include <stdio.h>
#include <fcntl.h>
#include <trap.h>
```

Example continued on the following page.



```
long count;
int keyboard, seconds = 0;
void handler (trap_no) int trap_no;
{
    char c;
    if (trap_no == 38) /* timer */
        seconds += 1;
    else /* CLEAR */
        /* Clear all key presses. */
        while (read (keyboard, &c, 1) > 0) ;
        printf("\fN=%ld SEC=%d", count, seconds);
        /* Reset counter and timer. */
        count = 0L;
        seconds = 0;
}
}
main ()
{
    struct trap_mask take;
    int slot;

    keyboard = open ("/dev/stdin", 0_RDONLY);
    /* Create mask for CLEAR key and 1 second timer */
    take = mask_or (mask_of (32), mask_of(38));
    slot = arm_guard(handler, take, take);

    printf("\fPRESS CLEAR");

    while (1)
        /* forever increment count. At 50,000 invoke CLEAR handler */
        if (50000L == ++count)
            dispatch_guard(mask_of(32))
                /* Cause CLEAR key handler */
}
}
```

## \_free()

Behaves like the Standard C for TXO function `free()`, but performs additional error checking.

**Prototype** `#include <txostd.h>`  
`int _free( ptr );`  
`char *p;`

**Parameters** `ptr` is a pointer, allocated via `malloc()` or `realloc()`, to be returned to the free list.

**Returns** TRUE (non-zero) The item appears to have been freed successfully. There may be forms of heap corruption which are not detectable and are reported as TRUE even though the pointer may not have been freed.

FALSE (zero) The pointer `ptr` is null, appears to not have been allocated via `malloc()`, appears to have been freed previously, or the `_free()` operation has failed.

**Notes** Space returned by the `_free()` function is retained in the free list and in the heap, not returned to the system or stack.

The functions `_free()` and `free()` are identical except that `_free()` provides a return value while `free()` does not.

**Example** See following page.

Example

```
/* _FREE.C */
#include <stdio.h>
#include <memory.h>
#include <txostd.h>

char *p,*calloc();
main () {
    /* Allocate 20 bytes of memory. */
    p = calloc(20,1);
    /* Write to memory and display what was written. */
    memset(p,'X',16);
    printf("\f%s",p);
    /* Free the memory. Display message if there was a problem. */
    if (!_free(p)) printf("\fAlloc Error");
}
```

## file\_copy()

---

Copies a source file to a destination file.

**Prototype** #include <txstd.h>  
int file\_copy(from, to);  
char \*from;  
char \*to;

**Parameters** from and to are null-terminated file names.

**Returns** 0 Success.  
-1 Failure, with errno set to a specific error value.

**Note** This function is supported on EPROM Version 10 and higher.

**Example** See following page.

Example

```
/* FILE_COPY.C */
#include <config.h>
#include <stdio.h>
#include <txostd.h>
#include <txosvc.h>

char buffer[10];

main () {
    /* Make a copy of the file CONFIG.SYS */
    file_copy("CONFIG.SYS", "TEMP");

    /* Allow the user to edit the duplicate file. */
    write(STDOUT, "\fEDIT CONFIG.SYS", 16);
    getchar();
    SVC_STORE("TEMP");

    /* Ask the user if they want to save their edits. */
    write(STDOUT, "\fSAVE EDITS?", 12);
    SVC_KEY_TXT(buffer, 1, 1, 1);

    /* Save edits if desired. */
    if (buffer[1] == 'Y') {
        file_copy("TEMP", "CONFIG.SYS");
        write(STDOUT, "\fEDITS SAVED", 12); }
    else
        write(STDOUT, "\fEDITS NOT SAVED", 16);

    /* Delete the temporary file. */
    remove("TEMP");
}
```