390 DIAL
395 LAN
395 DIAL

getcontrast()

Returns the current contrast level on a pixel-type display.

Prototype

```
#include <dsp_90.h>
int getcontrast(void);
```

Parameters None.

Returns Level value, 0 to 15, with 0 showing the most contrast, and 15 showing the least contrast.

Notes Available only to pixel-type display terminals.

Related `setcontrast()`

Example

```
/*  CONTRAST.C  */

#include <stdio.h>
#include <txosvc.h>
#include <dsp_90.h>

char buffer[10];
int contrast;
main () {

   while(1) {
      setcontrast(0);
      do {
         printf("\f  Enter Contrast\n(Between 0 and 15)");
         SVC_KEY_TXT(buffer, 0, 2, 1);
         contrast = SVC_2INT(buffer);
      } while ( (contrast < 0) || (contrast > 15));
      setcontrast(contrast);
      contrast=getcontrast();
      printf("\fThe Contrast Is %2d  CLEAR To Reset", contrast);
      getchar();
   }
}
```

390 DIAL
395 LAN
395 DIAL

get_env()

Retrieves the value of an environment variable from the CONFIG.SYS file.

Prototype

```
#include <txostd.h>
int get_env( key, buffer, max_bytes );
char *key, *buffer;
int max_bytes;
```

Parameters

`key` is a null-terminated string identifying the environment variable.

`buffer` contains the contents of the environment variable.

`max_bytes` is the size of the buffer.

Returns

`bytes`	The number of bytes stored into `buffer`.
0	The environment variable does not exist.

Notes

`get_env()` does not store the terminating NULL character into the buffer. Use this function instead of `getenv()` to avoid decreasing heap space with each call.

Related

`put_env()`

Example

```
/* GET_ENV.C */

#include <stdio.h>
#include <txostd.h>

char buffer[10];
main () {

   /* Create a new variable in CONFIG.SYS */
   put_env("NEWKEY", "NEWVAL", 6);

   /* Get the value of 'NEWKEY'.
      Use return length to make NULL-terminated string. */
   buffer[get_env("NEWKEY", buffer, 10)]=0;

   /* Display value of 'NEWKEY' */
   printf("\fNEWKEY = %s",buffer);
}
```

getfont()

Returns the name of the Font Definition File for the current font setting on a pixel-type display.

Prototype

```
#include <dsp_90.h>
int getfont(char *font);
char *font;
```

Parameters `font` is the storage area for the Font Definition File name.

Returns The function's return value is always zero.

Notes Available only to pixel-type display terminals.

The size of the `font` parameter must be at least 13 bytes to accommodate the full name.

Example

```
/* FONT.C */

#include <stdio.h>
#include <dsp_90.h>

void main()
{
   char font[13];

   getfont(font);
   printf("\fFONT: %s", font);

   setfont(font);
}
```

390 DIAL
395 LAN
395 DIAL

getfontinfo()

Returns pixel-type display font ROM information.

Prototype

```
#include <dsp_90.h>
int = getfontinfo( index, buffer );
int  index;
char *buffer;
```

Parameters

`index` is the font page number in font ROM.

`buffer` is the storage area for font information.

Returns

0	Success.
-1	Failure. If `index` exceeds the number of font pages in the font ROM, an error is returned and `errno` is set to `ENOENT`.

Notes

Available only to pixel-type display terminals. Returns information about the font page at the specified `index` location in the font ROM. This parameter represents the sequential position of the font page within ROM, where the first font page is 0. The font page information is copied into `buffer`, which must be at least 24 bytes long.

This function returns information about the font ROM, not about the fonts or Font Definition Files used on the terminal via the `setfont()` and `resetdisplay()` functions.

The format for the information returned in `buffer` is as follows:

`char`	`rom_name[9]`	Null-terminated name of font ROM
`char`	`font_page[9]`	Null-terminated name of font page
`char`	`x`	Width (in pixels) of one character
`char`	`y`	Height (in pixels) of one character
`char`	`grid_id`	Grid setting of this font
`char`	`reserved[3];`	Unused

The application can make successive calls to this function to get information on all the font pages in the font ROM.

Example

```
/* FONTINFO.C */

#include <stdio.h>
#include <txosvc.h>
#include <dsp_90.h>

void main()
{
   char gfibuffer[24];
   int i, status;

   for(i=0; getfontinfo(i, gfibuffer) == 0; i++)
   {
      printf("\fPAGE NUMBER %d:", i);
      SVC_WAIT(10000);

      printf("\fROM NAME: %s", gfibuffer);
      SVC_WAIT(10000);

      printf("\fPAGE NAME: %s", gfibuffer + 9);
      SVC_WAIT(10000);

      printf("\fWIDTH: %d", (int)(gfibuffer[18]) );
      SVC_WAIT(10000);

      printf("\fHEIGHT: %d", (int)(gfibuffer[19]) );
      SVC_WAIT(10000);

      printf("\fGRID ID: %d", (int)(gfibuffer[20]) );
      SVC_WAIT(10000);
   }
   printf("\fDONE");
}
```

390 DIAL
395 LAN
395 DIAL

getgrid()

Returns current grid setting for the selected font (on pixel-type displays).

Prototype

```
#include <dsp_90.h>
grid_id = getgrid(void);
int grid_id;
```

Parameters None

Returns `grid_id` is a constant defined in `<device.h>` naming the grid setting.

Notes Available only to pixel-type display terminals.

Example

```
/* GETGRID.C */

#include <stdio.h>
#include <dsp_90.h>

main()
{

   printf("%d", getgrid());

}
```

getkey()

Retrieves the data associated with the specified key value from a keyed file.

Prototype

```
#include <txostd.h>

int getkey( key, buffer, max_bytes, file_name );

char *key, *buffer, *file_name;
int max_bytes;
```

Parameters `file_name` is a pointer to a null-terminated string identifying the file, `key` is a null-terminated string identifying the pair, `buffer` is a pointer to an array which will contain the contents of the variable, and `max_bytes` is the size of the buffer.

Returns

`bytes_read`	Number of bytes read into the buffer; zero means the key does not exist.
`-1`	File does not exist. `errno` set. Values for `errno` include: `EBADF`

Notes The file does not have to be open; an implied open and close is performed automatically by this function. Thus, a `getkey()` call can be used successfully even when the maximum number of opened files has been previously reached.

Related `putkey()`

Example See following page.

Example

```
/* KEY.C */

#include <stdio.h>
#include <fcntl.h>
#include <txostd.h>

char buffer[10];
int file_handle;
main () {

   /* File must exist, but doesn't need to be open. */
   file_handle=open("NEWFILE",O_CREAT);
   close(file_handle);

   /* Create a new variable in NEWFILE */
   putkey("KEY", "VALUE", 5, "NEWFILE");

   /* Get the value of KEY
      Use return code to make NULL-terminated string. */
   buffer[getkey("KEY", buffer, 10, "NEWFILE")]=0;

   /* Display the value of KEY */
   printf("\fKEY = %s",buffer);
}
```

get_lan_config()

Retrieves the current parameters used to configure the LAN.

Prototype

```
#include <lan.h>
int get_lan_config ( handle, term_parms,
                     prot_parms );
int handle;
LAN_TERM_PARMS *term_parms;
LAN_PROT_PARMS *port_parms;
```

Parameters

`handle` is the value returned by `open()`.

Returns

0	Success
-1	Failure, with `errno` set to a specific error value (if unable to access the underlying LAN configuration parameters). Also returns the values of these parameters in the data structures `term_parms` and `prot_parms`.

Notes

The data structure `LAN_TERM_PARMS` (in `<lan.h>`) is as follows:

```
typedef struct
{       unsigned char receive_timeout;
        unsigned char resource_definition;
        unsigned char transmit_retries;
        unsigned char throttle;
} LAN_TERM_PARMS;
```

`receive_timeout` specifies the number of seconds (1–10) that the terminal waits for an ACK before the transmission attempt fails.

`resource_definition` is for application use, e.g., to identify specific LAN resources, such as printer server or host gateway.

`transmit_retries` specifies the number of additional attempts (1 to 10) to transmit an unACKed message, e.g., `transmit_retries=3` means a total of four tries.

`throttle` is usually on (1) in most terminals on the LAN, preventing low address terminals from hogging the LAN

under heavy loads, and off (0) for terminals needing better LAN throughput (gateway or server terminals).

The data structure LAN_PROT_PARMS (in <lan.h>) is as follows:

```
typedef struct
{       unsigned char rate;
        unsigned char address;
        unsigned char highest_address;
        unsigned char timing_window;
} LAN_PROT_PARMS;
```

rate is LAN baud rate, determined by *LBR in CONFIG.SYS (must match for all terminals on the LAN).

address is the terminal's LAN address, determined by *LAD in CONFIG.SYS.

highest_address is the highest expected address on the LAN, determined by *LHA in CONFIG.SYS (must match in all terminals on the LAN).

timing_window is either 1 (fast) or 2 (slow), as determined by the *LTW entry in CONFIG.SYS. This value must match in all terminals on the LAN. Fast setting (normal) increases LAN throughput. Slow setting is for LANs with electrical interference problems.

Related put_lan_config()

Example

```
/* GETLCFG.C */

#include <lan.h>
LAN_TERM_PARMS term_parms;
LAN_PROT_PARMS prot_parms;
main()
{
   /* Retrieve LAN configuration parameters. */
   get_lan_config (hLAN, &term_parms, &prot_parms);

   /* Display the parameters. */
   printf("\fRCV TMOUT = %d", term_parms.receive_timeout);
   printf("\fRES_DEFN = %d", term_parms.resource_definition);
   printf("\fXMT RETRY = %d", term_parms.transmit_retries);
   printf("\fTHROTTLE = %d", term_parms.throttle);
   printf("\fBAUD RT = %d", prot_parms.rate);
   printf("\fTERM_ADDR = %d", prot_parms.address);
   printf("\fHI_ADDR = %d", prot_parms.highest_address);
   printf("\fTIME_WNDW = %d", prot_parms.timing_window);
}
```

getscrollmode()

Returns the current settings for display scrolling.

Prototype

```
#include <txostd.h>
int getscrollmode (void);
```

Parameters None.

Returns The current scroll mode is returned as follows:

- 0 Scrolling disabled
- 1 Horizontal scrolling enabled
- 2 Vertical scrolling enabled

Related `setscrollmode()`

Example

```
/* SCROLL.C */

#include <stdio.h>
#include <txostd.h>

main () {

   /* Fill display. */
   printf("\f>>>>>>>><<<<<<<<");

   /* Define window and move cursor to the last position. */
   window(3, 1, 14, 1);

   /* Display in the default mode, rightmost portion should remain. */
   /* Note that the default mode is identical to mode 1.           */
   /* >>NT MODE IS 1<< */
   printf("\fTHE CURRENT MODE IS %d", getscrollmode());
   getchar();

   /* Mode 0: Should not slide off the left edge of the window. */
   setscrollmode(0);
   /* >>0 IS THE CUR<< */
   printf("\f%d IS THE CURRENT MODE", getscrollmode());
   getchar();

   /* Mode 1: Should slide off the left edge of the window. */
   setscrollmode(1);
   /* >>NT MODE IS 1<< */
   printf("\fTHE CURRENT MODE IS %d", getscrollmode());
   getchar();

   /* Mode 2: When a character is written beyond the edge of the */
   /*         window, the current lines are scrolled up one line */
   /*         and the cursor is moved to the beginning of the    */
   /*         last line in the window.                           */
   setscrollmode(2);
   /* >>MODE IS 2   << */
   printf("\fTHE CURRENT MODE IS %d", getscrollmode());
}
```

gotoxy()

Moves the cursor to a specified position within a display window defined by the `window()` function.

Prototype

```
#include <txostd.h>
void gotoxy( x, y );
int x;
int y;
```

Parameters `x` and `y` are the horizontal and vertical coordinates, respectively. If `x` or `y` exceed the dimensions of the current window, then minimum/maximum positions are used.

Returns VOID

Notes If a window is defined by the coordinates (6,1,8,2), calling `gotoxy(2,1)` places the cursor at physical location 7,1.

Pixel-type display: This function has the restriction that the cursor cannot be repositioned in the middle of an existing double-wide character. This restriction is not enforced by the firmware, but if violated may result in undefined behavior.

Related `clreol()`, `window()`

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();
}
```

insert()

Inserts data from a buffer into a file at the current file position pointer. All subsequent data in the file is moved to make room for the new data.

Prototype

```
#include <io.h>
int insert( handle, buffer, size );
int handle, buffersize;
char *buffer
```

Parameters `handle` is the value returned by `open()`, `buffer` is a pointer to an area of memory and `size` is the number of bytes to insert.

Returns

`bytes`	The number of bytes inserted.

`-1`	Failure, with `errno` set to one of the following values:
	`EBADF`
	`ENOSPC`

Notes `insert()` operates very much like `write()`. In fact, if the seek pointer is at the end of the file, the two operations are identical.

Example

```
/* FILEOPS.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", O_CREAT+O_RDWR);

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

   /* Seek to the beginning of the word "World" */
   lseek(file_handle, 6, SEEK_SET);

   /* Insert "Big" to make "Hello Big World" */
   insert(file_handle, "Big ", 4);

   /* Read from the file and display the result. */
   lseek(file_handle, 0, SEEK_SET);
   read(file_handle, buffer, 15);
   buffer[15] = 0;
   printf("\f%s",buffer);

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

insert_cvlr()

Inserts a compressed variable-length record into a file.

Prototype

```
#include <vlr.h>
int insert_cvlr( handle, buffer, size );
int handle, size;
char *buffer;
```

Parameters `handle` is the value returned by `open()`, `buffer` is a pointer to an area of memory and `size` is the number of bytes to insert.

Returns

`bytes_written`	The number of bytes written to the file.

`-1`	Failure, with `errno` providing a specific error value. Values for `errno` include:
	`EBADF`
	`EINVAL`
	`ENOSPC`

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 successfully insert the record into the file, the pointer must be correctly placed.

❖ *Warning: Data values cannot be outside the range* `0x00-0x5F`. *These values are not correctly translated during the data compression process. See Chapter 6, File Management.*

Example

```
/* CVLR.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", O_CREAT+O_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);
}
```

insert_vlr()

Inserts a variable-length record into a file.

Prototype

```
#include <vlr.h>
int insert_vlr( handle, buffer, size );
int handle, size;
char *buffer;
```

Parameters `handle` is the value returned by `open()`, `buffer` is a pointer to an area of memory and `size` is the number of bytes to insert.

Returns

`bytes_written`	The number of bytes written to the file.

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

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 successfully insert the record into the file, the pointer must be correctly placed.

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", O_CREAT+O_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);
}
```

insline()

Inserts a blank line before the display line containing the cursor.

Prototype

```
#include <txostd.h>
void insline(void);
```

Parameters None

Returns VOID

Notes All lines below are moved down one line, thus creating a blank line on the display. The cursor position is not changed.

Related delline()

Example

```
/* INSLINE.C */

#include <stdio.h>
#include <txostd.h>

main () {
   /* Fill the display and wait. */
   printf("\f1234567890ABCDEF");
   getchar();

   /* Clear the screen using insline. */
   insline();
}
```

ioctl()

Used to perform device-level functions and obtain file or device-related information.

Prototype

```
#include <io.h>

int ioctl( handle, type, buffer );

int handle;
unsigned int type;
void *buffer;
```

Parameters

`handle` is the device/file handle returned by `open()`.

`buffer` is a pointer to a region of memory.

`type` is an unsigned integer value that varies depending on the device being used and the desired result. Varying amounts of data may be written to the buffer depending on the `type` parameter and the kind of device being accessed.

Returns

Returned values are file and device-specific. See file and device-specific chapters of this manual for details.

ioctlc()

Prototype

```
#include <io.h>

int ioctlc( handle, type, buffer, length );

int handle, length;
unsigned int type;
void *buffer;
```

Parameters

Same as `ioctl()` above with the inclusion of the `length` parameter, which specifies the size of the buffer.

Notes

`ioctlc()` is normally used to read the next message in the LAN reject queue (using `GetCtrl | 2`). This `ioctlc()` call returns 0 on success, or -1 on error with `errno` set to either `EINVAL` if the command was invalid, or `EBADF` if the LAN is not open or is not initialized.

lseek()

Moves the file position pointer to a new location in a file.

Prototype

```
#include <io.h>
long lseek( handle, offset, origin );
int handle, origin;
long offset;
```

Parameters

`handle` is the file handle returned by `open()`.

`offset` is a long word specifying the number of bytes.

`origin` is one of the following constants:

`SEEK_SET` – Seek from the beginning of the file.
`SEEK_CUR` – Seek from current position of the file pointer.
`SEEK_END` – Seek from the end of the file.

An offset of zero from `SEEK_SET` places the file pointer before the first byte in the file, and an offset of one from `SEEK_SET` places the file pointer after the first byte in the file.

Returns

`pointer`	The new value of the file position pointer is returned upon successful operation. This is always the total number of bytes from the beginning of the file.
`-1`	Failure, with `errno` set to a specific error value. The file position pointer is not changed. Values for `errno` include: `EBADF` `EINVAL`

The following call:

```
new_position = lseek(handle,0L,SEEK_END);
```

returns the size of the data portion of the file. The file position pointer is pointing to the end of the file.

Notes

Although `offset` is a signed long integer, only positive offsets are meaningful when seeking from the beginning of the file; only negative offsets are meaningful when seeking from the end of the file.

Related

`seek_cvlr()`, `seek_vlr()`

Example

```
/* FILEOPS.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", O_CREAT+O_RDWR);

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

   /* Seek to the beginning of the word "World" */
   lseek(file_handle, 6, SEEK_SET);

   /* Insert "Big" to make "Hello Big World" */
   insert(file_handle, "Big ", 4);

   /* Read from the file and display the result. */
   lseek(file_handle, 0, SEEK_SET);
   read(file_handle, buffer, 15);
   buffer[15] = 0;
   printf("\f%s",buffer);

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

mask_and()

Performs a logical AND operation on two trap masks.

Prototype

```
#include <trap.h>
struct trap_mask mask_and( a, b );
struct trap_mask a,b;
```

Parameters a and b are the trap masks to AND together

Returns The trap mask result of ANDing a and b.

Example

```
/* MASK_AND.C */

#include <stdio.h>
#include <trap.h>
#include <txostd.h>

/* This routine will print the 64 bit mask in 16 hex digits.*/
/* The mask will read with bit 63 on the left and 0 on the
   right.                                                    */
void print_mask(mask)
struct trap_mask mask;
{ int i;
  clrscr();
  for (i=7; i>=0; i--) printf("%02x",mask.a[i]);
  getchar();
}

struct trap_mask mask1,mask2,mask3;
main () {
   /* 'And' a mask with bits 0 and 63 on, with a mask with
      just bit 63 on.  Result is mask with just bit 63 on.  */
   /* Display: 8000000000000000 */
   mask1 = mask_of(0);
   mask2 = mask_of(63);
   mask3 = mask_or(mask1,mask2);
   mask1 = mask_and(mask2,mask3);
   print_mask(mask1);
}
```

mask_clear()

Resets the specified bits in a trap mask.

Prototype

```
#include <trap.h>
struct trap_mask mask_clear( a, b );
struct trap_mask a,b;
```

Parameters

a is the trap mask to be modified,
b is a mask identifying the bits in mask a to be reset.

Returns

A mask of a bits, with b bits excluded.

Example

```
/* MASK_CLR.C */

#include <stdio.h>
#include <trap.h>
#include <txostd.h>

/* Prints the 64 bit mask in 16 hex digits.          */
/* The mask will read with bit 63 on the left        */
/* and 0 on the right.                               */
void print_mask(mask)
struct trap_mask mask;
{ int i;
  clrscr();
  for (i=7; i>=0; i--) printf("%02x",mask.a[i]);
  getchar();
}

struct trap_mask mask1,mask2;
main () {
   /* Mask1 has bits 0 and 1 set.  Mask2 has bit 0 set.     */
   /* Mask_Clear will clear all bits in Mask1 that are set  */
   /* in Mask2.  The result leaves only bit 1 in Mask1 set. */
   /* Display: 00000000000002 */
   mask1 = mask_of(0);
   mask1 = mask_of(1);
   mask2 = mask_of(0);
   mask1 = mask_clear(mask1,mask2);
   print_mask(mask1);
}
```

mask_empty()

Resets all bits in a trap mask to 0.

Prototype

```
#include <trap.h>
struct trap_mask mask_empty( void );
```

Parameters None

Returns A mask with all bit positions set to 0.

Example

```
/* MASK_EMP.C */

#include <stdio.h>
#include <trap.h>
#include <txostd.h>

/* This routine will print the 64 bit mask in 16 hex digits.*/
/* The mask will read with bit 63 on the left and 0 on the
   right.                                                    */
void print_mask(mask)
struct trap_mask mask;
{ int i;
  clrscr();
  for (i=7; i>=0; i--) printf("%02x",mask.a[i]);
  getchar();
}

struct trap_mask mask1;
main () {
   /* This routine will clear all 64 bits. */
   /* Display: 0000000000000000 */
   mask1 = mask_empty();
   print_mask(mask1);
}
```

mask_fill()

Sets all bits in a trap mask to 1.

Prototype

```
#include <trap.h>
struct trap_mask mask_fill( void );
```

Parameters None

Returns A mask with all bit positions set to 1.

Example

```
/* MASK_FIL.C */

#include <stdio.h>
#include <trap.h>
#include <txostd.h>

/* This routine will print the 64 bit mask in 16 hex digits.*/
/* The mask will read with bit 63 on the left and 0 on the
   right.                                                   */
void print_mask(mask)
struct trap_mask mask;
{ int i;
  clrscr();
  for (i=7; i>=0; i--) printf("%02x",mask.a[i]);
  getchar();
}

struct trap_mask mask1;
main () {
   /* Set all 64 bits.  */
   /* Display: FFFFFFFFFFFFFFFF */
   mask1 = mask_fill();
   print_mask(mask1);
}
```

mask_in()

Performs a bit test on a trap mask to determine if the specified bit is on or off.

Prototype

```
#include <trap.h>
int mask_in( a, k );
struct trap_mask a;
int k;
```

Parameters

a is trap mask to test.
k is the bit number to test in a.

Returns

TRUE	(1)	The specified bit is on.
FALSE	(0)	The bit is off.

Example

```
/* MASK_IN.C */

#include <stdio.h>
#include <trap.h>
struct trap_mask mask1;
main () {
   /* Set bit 33 in Mask1.  Use Mask_In to test if bit 33 */
   /* is set (true), next test if bit 19 is set (false).  */
   /* Display: 1=TRUE   0=FALSE */
   mask1 = mask_of(33);
   printf("\f%d=TRUE   %d=FALSE", mask_in(mask1,33),
          mask_in(mask1,19));
}
```

mask_of()

Use this function to set a bit in a trap mask to identify a "trappable" event. Bits are integer values of 0 – 63 corresponding to the trap numbers defined in <trap.h> and described in Chapter 7, TXO Exception Handling.

Prototype

```
#include <trap.h>
struct trap_mask mask_of( k );
int k;
```

Parameters k is the bit number to be set.

Returns A trap mask with the k bit set (if k is a value between 0 and 63).

Example

```
/* MASK_OF.C */

#include <stdio.h>
#include <trap.h>
#include <txostd.h>

/* This routine will print the 64 bit mask in 16 hex digits.*/
/* The mask will read with bit 63 on the left and 0 on the
   right.                                                    */
void print_mask(mask)
struct trap_mask mask;
{ int i;
  clrscr();
  for (i=7; i>=0; i--) printf("%02x",mask.a[i]);
  getchar();
}

struct trap_mask mask1;
main () {
   /* Set mask bit 0 */
   /* Display: 0000000000000001 */
   mask1 = mask_of(0);
   print_mask(mask1);
}
```

mask_or()

Performs a logical OR operation on two trap masks.

Prototype

```
#include <trap.h>
struct trap_mask mask_or( a, b );
struct trap_mask a,b;
```

Parameters a and b are the trap masks to OR together.

Returns The ORed value of trap masks a and b.

Example

```
/* MASK_OR.C */

#include <stdio.h>
#include <trap.h>
#include <txostd.h>

/* This routine will print the 64 bit mask in 16 hex digits.*/
/* The mask will read with bit 63 on the left and 0 on the
   right.                                                    */
void print_mask(mask)
struct trap_mask mask;
{ int i;
  clrscr();
  for (i=7; i>=0; i--) printf("%02x",mask.a[i]);
  getchar();
}

struct trap_mask mask1,mask2,mask3;
main () {
   /* Or masks of bit 0 and 63 together. */
   /* Display: 8000000000000001 */
   mask1 = mask_of(0);
   mask2 = mask_of(63);
   mask3 = mask_or(mask1,mask2);
   print_mask(mask3);
}
```

open()

Prepares a file or device for subsequent processing.

Prototype

```
#include <io.h>
int open( file_name, attributes );
char *file_name;
int attributes;
```

Parameters

`file_name` is a pointer to a null-terminated string containing a file or device name.

If a device is being opened, the `attributes` argument will vary according to the device; refer to the section covering the particular device in question. Otherwise `attributes` may specify one or more of the following constants (defined in `<fcntl.h>`):

`O_RDONLY`	Open file for reading only.
`O_WRONLY`	Open file for writing only.
`O_RDWR`	Open file for both reading and writing.
`O_APPEND`	Open file with file pointer initially set to EOF. The file position pointer will always be placed at the end of the file before a write occurs.
`O_CREAT`	Creating a new file for writing. If the file already exists, it is simply opened with write access.
`O_TRUNC`	Open and truncate an existing file. The file must exist; its content, if any, is deleted.
`O_EXCL`	Used with `O_CREAT` to return an error value if the file already exists.
`O_CODEFILE`	Used with `O_CREAT` and `O_WRONLY` (or `O_RDWR`) to open what will be a new, dynamically-created code (or executable) file. This attribute is not required to open an existing code file.

Returns

handle	If successful, open() returns a file handle (16-bit integer) used in all subsequent file operations.
-1	Failure, with errno set to a specific error value. Values for errno include: EMFILE EEXIST ENOENT ENOSPC

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", O_CREAT+O_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", O_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);
}
```

open_codefile()

Opens a file as an executable code file.

Prototype

```
#include <io.h>
int open_codefile( file_name, attributes, length );
char *file_name;
int attributes;
unsigned long length;
```

Parameters

`file_name` and `attributes` are identical to the `open()` function's parameters.

Be aware that the `length` parameter is not used on the 300 Series platform.

Returns

`handle`	If successful, `open_codefile()` returns a file handle used in all subsequent file operations.

`-1`	Failure, with `errno` set to a specific error value. Values for `errno` include:
	`EMFILE`
	`EEXIST`
	`ENOENT`
	`ENOSPC`

Notes

This function is identical to using `open()` with the `attribute` parameter ORed with `O_CODEFILE` (e.g., `attribute = O_RDONLY + O_CODEFILE`).

Example

See following page.

Example

```
/* OPENCODE.C */

#include <stdio.h>
#include <io.h>
#include <fcntl.h>
#include <string.h>
#include <txostd.h>
#include <txosvc.h>

int old_codefile, new_codefile, bytes;
char filename[40], buffer[300];

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

   /* Get the name of the file currently running. */
   strcpy (filename, argv[argc-1]);

   /* See if it was a recreation of the original. */
   if (!strcmp(filename, "NEW-CODEFILE")) {
      printf("\fCODEFILE COPIED!");
      while (1);
   }

   /* Open the existing codefile and create a new codefile.       */
   /* The way to open the file using open() is shown in comments. */
/* old_codefile = open(filename, O_RDONLY + O_CODEFILE); */
   old_codefile = open_codefile(filename,
                                O_RDONLY, (unsigned long)0);
/* new_codefile = open("NEW-CODEFILE",
                       O_CREAT + O_WRONLY + O_CODEFILE); */
   new_codefile = open_codefile("NEW-CODEFILE",
                                O_CREAT + O_WRONLY,
                                (unsigned long)0);

   /* Make a copy of the current codefile. */
   do { bytes = read(old_codefile, buffer, 240);
        write(new_codefile, buffer, bytes);
      } while (bytes);

   /* Restart the new codefile. */
   SVC_RESTART("NEW-CODEFILE");
}
```

pending_traps()

Copies a list of the current pending events/exceptions, and resets any or all pending events/exceptions to avoid their activation.

Prototype

```
#include <trap.h>
void pending_traps( current_mask, clear_mask );
struct trap_mask *current_mask, *clear_mask;
```

Parameters See Notes below.

Returns VOID

Notes If `current_mask` is not NULL, a copy of the currently pending system mask is copied into the area pointed to by `current_mask`.

If `clear_mask` is not NULL, then for every bit on in `clear_mask`, the corresponding bit in the pending mask is reset.

If both `current_mask` and `clear_mask` are non-NULL pointers, the effect of copying and updating the current mask is performed as an operation that is indivisible at the system and device level. Since no window of time exists between these operations, a device can never post a request that would be immediately reset.

Example

```
/* Get pending traps from system                */
 pending_traps( &pending, NULL );
/* Separate user traps from system traps        */
/* Get bits in pending, not in user trap mask   */
sys_pend = mask_clear( pending, usr_mask );
```