Example

```
/* STORE.C */


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

main() {

   /* Editing CONFIG.SYS */
   printf("\fEDIT CONFIG.SYS");
   getchar();
   SVC_STORE("CONFIG.SYS");

   /* Create a new file and edit it. */
   printf("\fEDIT NEW.FILE");
   getchar();
   SVC_STORE("NEW.FILE");

   /* Go back and re-edit CONFIG.SYS */
   /* File name is not case sensitive */
   printf("\fCONFIG.SYS AGAIN");
   getchar();
   SVC_STORE("CoNfIG.Sys");

   /* Re-edit NEW.FILE */
   /* File name is not case sensitive */
   printf("\fNEW.FILE AGAIN");
   getchar();
   SVC_STORE("NeW.fIlE");

   /* End of SVC_STORE() example */
   printf("\fEND EXAMPLE");

}
```

SVC_TICKS()

Allows the user to check if a tick value has expired, or to read the system's tick counter.

Prototype

```
#include <txosvc.h>
int SVC_TICKS ( action, longadr );
int action;
long *longadr;
```

Parameters The `action` parameter determines the behavior of the function.

- `0` Compare the specified longword value against the system's current longword "tick" counter. The return value indicates whether or not the user's value has "expired" (0=expired, 1=pending).
- `1` Copy the system's current longword "tick" counter to the caller's longword.

Returns Return values are based on the value of `action`:

action =	Returned Value
0	0 = User's tick as expired. 1 = User's tick is pending (has not expired).
1	Always returns 1.

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

Related `tick_compare()`

Example See following page.

Example

```
/* TICKS.C */

#include <config.h>
long t1;

main () {
    write(STDOUT, "\fWAIT 3 SECONDS...", 18);

    /* Get the current tick count. */
    SVC_TICKS (1, &t1);

    /* Set the count to be 3 seconds into the future. */
    t1 += 3 * TICKS_PER_SEC;

    /* Wait for the tick value to expire. */
    while(SVC_TICKS (0, &t1));

    write(STDOUT, "\fTIME'S UP!", 11);
}
```

SVC_2INT()

Converts an ASCII decimal number to a binary integer.

Prototype

```
#include <txosvc.h>
unsigned int SVC_2INT( source );
char *source;
```

Parameters

`source` is a counted string containing the ASCII decimal number to convert.

Notes

The `source` string is treated as an unsigned decimal number which is converted to an unsigned binary integer. Non-numeric characters are ignored. A zero-length string is converted to zero.

For example, the string:

```
<0x18>"-0+0-*2**2**2*CRAZY8S"
```

would be read "002228" and converted to 0x08B4. Note the first byte contains the length of the string (e.g., the count byte).

`SVC_2INT()` does not check for overflow conditions. Attempting to convert numbers above 65535 will produce erroneous results.

Example

```
/* 2INT.C */

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

char source[25];
main() {

   /* Build a counted string. */
   strcpy(&source[1],"-0+0-*2**2**2*CRAZY8S");
   *source=strlen(&source[1]) + 1;

   /* Convert the string and display in decimal and hex. */
   printf("\f%06d %04x",SVC_2INT(source),SVC_2INT(source));
}
```

SVC_UNPK4()

Decompresses the output of SVC_PACK4() to recover the original text.

Prototype

```
#include <txostd.h>
unsigned int SVC_UNPK4( dest, source, size );
char *dest, *source;
unsigned int size;
```

Parameters

dest passes the address of a buffer to receive the decompressed data (i.e., at least twice as large as the size parameter).

source passes the address of a buffer containing the compressed data.

size specifies the number of bytes (in source) to be converted. The value of size must be less than or equal to 255.

Returns

bytes_placed	The number of decompressed bytes placed in the destination buffer.
-1	Failure. size is too large.

Notes

This routine complements the SVC_PACK4() routine; it decompresses the output of SVC_PACK4() to recover the original text, as long as the original text is in the required range. It is similar to the read_cvlr() routine for reading from memory-based files.

Related

SVC_PACK4(), read_cvlr()

Example

```
/* UNPK4.C */

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

char src[20],dest[20],result[20];
main() {

   /* Build an alphanumeric string */
   strcpy(src, "1234567890ABCDEF");

   /* Pack the string, unpack it, then print it. */
   SVC_PACK4(dest, src, strlen(src));
   result[SVC_UNPK4(result, dest, strlen(dest))] = 0;
   printf("\f%s",result);
}
```

SVC_VERSION()

Returns the terminal's EPROM identification string.

Prototype

```
#include <txosvc.h>
void SVC_VERSION( buffer );
char *buffer;
```

Parameters The firmware version is returned in `buffer` as a counted string. A maximum of nine bytes is returned.

Returns VOID

Related Same as `SVC_INFO_EPROM()`

Example

```
/* VERSION.C */

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

char buffer[10];
main() {

   /* Get the EPROM ID. */
   SVC_VERSION(buffer);

   /* Null terminate the string and display. */
   buffer[buffer[0]]=0;
   printf("\fVERSION %s",&buffer[1]);
}
```

SVC_WAIT()

Causes a software delay for a specified period of time.

Prototype

```
#include <txosvc.h>
void SVC_WAIT( time );
int time;
```

Parameters `time` is the number of milliseconds to delay.

Returns VOID

Notes This function executes a foreground loop until the approximate number of milliseconds specified by `time` has elapsed. `time` must be between 10 and 65535 milliseconds.

This function is not interruptable via traps and guards, and it should not be used for exact timing.

❖ *When ported to an OMNI 400 Series platform (i.e., multi-tasking environment), this function forces a task switch when* `time` *equals zero.*

Example

```
/* WAIT.C */

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

main() {
   printf("\fWAIT 10 SECONDS");
   SVC_WAIT(10000);
   printf("\fDONE!");
}
```

380 DIAL
385 DIAL
390 DIAL
395 DIAL

SVC_ZONTALK() *Modem Downloads*

Executes a modem-based ZONTALK 2000 download.

Prototype

```
#include <txosvc.h>
int SVC_ZONTALK(type);
int type;
```

Parameters

`type` takes the values:

`'F'`	Full Download
`'P'`	Partial Download
`'p'`	Partial Download and Continue
`NO_DIAL`	Modem already connected to host

Full download

All application files are transferred from the host download computer to the terminal. Once the download is complete, the application is started.

Partial download and start

Adds or replaces code or data files to the terminal file system. Once completed, the existing application is restarted.

Partial download and continue

The existing application resumes execution immediately following the `SVC_ZONTALK()` call.

NO_DIAL modifier

Can be added to the above parameters. Used if the modem is already opened and connected to the host computer. The terminal waits for an ENQ from the host, and then begins the download. Example:

```
status = SVC_ZONTALK( 'p' + NO_DIAL);
```

- If any of the *ZA, *ZP or *ZT entries are not set in CONFIG.SYS —and NO_DIAL is not specified—this function prompts the user to enter the missing values. These values are not checked if the NO_DIAL modifier is used. However, if *ZA and *ZT are not set, and the connected host is a true ZONTALK host, an error is returned.

Returns This call will not return when 'F' or 'P' are used and the download is successful, as the system is restarted.

When 'p' is used:

0 = Download successful.

-1 = Error. Possible causes are:

- parameter other than 'F', 'P' or 'p' used,
- an error occurred while the modem was attempting to connect (NO_DIAL not used); possible reasons for this are: NO LINE, NO ANSWER, NO CARRIER or BUSY,
- *ZA or *ZT entries are incorrect,
- an error occurred during a partial download and continue operation.

If an error occurs during an actual file download using either 'F' or 'P', the state of the system is undefined.

Example

```
/* ZONTALK.C */

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

main () {

   /* Assign ZONTALK parameters.                */
   /* If unassigned the user will be prompted. */
   put_env("*ZA", "ZTTEST", 6 );
   put_env("*ZT", "1", 1 );
   put_env("*ZP", "3392", 4 );

   /* Initiate a full download. */
   SVC_ZONTALK('F');
}
```

385 LAN
395 LAN

SVC_ZONTALK()

LAN Downloads

Application running on LAN terminal requests a file download from a server terminal. See *Chapter 10* for details.

Prototype

```
#include <txosvc.h>
#include <lan.h>

int SVC_ZONTALK(type);
int type;
```

Parameters

FULL_DL	Full download and restart application immediately after download completes.
PARTIAL_DL	Partial download and restart application immediately after download completes.
PART_DL_CONT	Partial download and continue—after download completes, control returns to the application at the point the call was made.
REJECT_DL	Do not perform download at this time—EOT packet is sent to download server signaling that the download request was refused.

The following flags may be added (or logical OR'ed) to any of the above types:

NO_DIAL	Do not open or configure the port, do not send Download Request packet.
LAN_DOWNLOAD	Use LAN port instead of MODEM port.

Returns

Ordinarily, SVC_ZONTALK() does not return, since the terminal is restarted when the download is completed. Control returns to the application only if the type parameter is set to PART_DL_CONT or to REJECT_DL. On success, SVC_ZONTALK() returns 0.

On error, SVC_ZONTALK() returns -1 in result and errno is set to one the following:

EINVAL	Invalid parameter setting (e.g., if the user requests a modem download)
EBADF	LAN not open or not configured and NO_DIAL flag was set

Notes Application-initiated LAN downloading is covered in detail in *Chapter 10, OMNI Peer-to-Peer LAN.* SVC_ZONTALK() for LAN downloading is included here for quick reference to the prototype and return values.

tick_compare()

Compares two timer values, reporting if the current "tick" value exceeds the ending "tick" value.

Prototype

```
#include <device.h>
int tick_compare ( cur_tick, end_tick );
long cur_tick, end_tick;
```

Parameters The tick value in `cur_tick` is compared to the value in `end_tick`.

Returns If `cur_tick` exceeds `end_tick`, zero is returned; otherwise 1 is returned.

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

Related `SVC_TICK()`

Example

```
/* TCKCMP.C */

#include <io.h>
#include <config.h>
#include <device.h>
#include <txosvc.h>
long t1;
long t2;

main () {

   /* Compute a tick value three seconds into the future. */
   SVC_TICKS(1,&t1);
   t1 += 3 * TICKS_PER_SEC;

   write(STDOUT, "\fWAIT 3 SECONDS...", 18);

   /* Loop until the current tick value exceeds the value computed above. */
   do
      SVC_TICKS(1, &t2); /* Get the current tick value. */
   while (tick_compare(t2, t1));

   write(STDOUT, "\fTIME'S UP!", 11);
}
```

trap_of()

Returns the trap number corresponding to the device and type of trap event specified.

Prototype

```
#include <trap.h>
int trap_of ( handle, whichtrap );
unsigned int handle, whichtrap;
```

Parameters

`handle` must be the handle of an open device.

`whichtrap` specifies the type of trap to get:

0	Normal Event
1	Exception Event
2	Modem Response Available (modem only)
3	Loss Of Carrier (modem only)

Returns

`trap_number`	The number of the trap requested for the specified device.
-1	Invalid settings for `handle` or `whichtrap`.

Example

```
/* TRAP_OF.C */

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

unsigned int modem;

main() {
   /* Open the modem. */
   modem = open("/dev/com4", 0);

   /* Display the trap numbers for the various types modem
traps. */
   printf("\fNORMAL TRAP = %d", trap_of(modem, 0)); getchar();
   printf("\fEXCEPTION = %d",   trap_of(modem, 1)); getchar();
   printf("\fRESP AVAIL = %d",  trap_of(modem, 2)); getchar();
   printf("\fNO CARRIER = %d",  trap_of(modem, 3));
}
```

unlink()

Deletes the file specified by *pathname*.

Prototype

```
#include <stdio.h>
int unlink(pathname);
char *pathname;
```

Parameters

pathname	Path name of file to be removed.

Returns Zero if the file is successfully deleted, or -1 if an error occurs.

Related close(), remove()

Example

```
/* UNLINK.C */

#include <stdio.h>
#include <fcntl.h>
int file_handle;

main()
{
  /* Create a file */
  file_handle = open("Test.Dat", O_CREAT+O_RDWR);
  close(file_handle);

  /* Unlink the file */
  unlink("Test.Dat");

  /* Make sure it is gone */
  if (open("Test.Dat", O_RDONLY) == -1)
    printf("\fFILE FOUND");
}
```

wherecur()

Writes to x and y values the current cursor position relative to the physical display (not the current window).

Prototype

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

Parameters

x	Horizontal cursor coordinate.
y	Vertical cursor coordinate.

Returns VOID

Notes If the last character written is at the last position of the window, the cursor remains at the last window position.

It is not possible for the cursor to be reported as being outside of the current window.

Example

```
/* WHERECUR.C */

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

int x,y;
main () {

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

   /* Define window, cursor should be placed in col 3 */
   window(3, 1, 14, 1);

   /* Get current cursor position. */
   wherecur(&x, &y);

   /* Clear only the window and display cursor col. */
   printf("\fSTART COL %d",x);
}
```

wherewin()

Writes to x and y values the current display window coordinates into the four integer variables.

Prototype

```
#include <txostd.h>
void wherewin( x1, y1, x2, y2 );
int *x1;
int *y1;
int *x2;
int *y2;
```

Parameters

x1	Upper left column coordinate.
y1	Upper left row coordinate.
x2	Lower right column coordinate.
y2	Lower right row coordinate.

Returns The current display window coordinates are copied into the four integer variables, where x1 and y1 are the column and row of the upper left corner of the window, and x2 and y2 are the column and row of its lower right corner.

Example

```
/* WINDOW.C */

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

int x1,y1,x2,y2;
main () {

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

   /* Define a window. */
   window(3, 1, 14, 1);

   /* Get window coordinates. */
   wherewin(&x1, &y1, &x2, &y2);

   /* Clear only the window and display window coordinates. */
   printf("\f(%d,%d) - (%d,%d)",x1, y1, x2, y2);
}
```

wherewincur()

Writes to x and y values the current cursor position relative to the current window (not the physical display).

Prototype

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

Parameters

x Horizontal cursor coordinate.

y Vertical cursor coordinate.

Returns The current display cursor position is copied into the two integer variables x and y, where x is the column and y is the row.

Notes If the last character written is at the last position of the window, the cursor remains at the last window position.

It is not possible for the cursor to be reported as being outside of the current window.

Example

```
/* WHWINCUR.C */

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

int x,y;
main () {

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

   /* Define window and move cursor to the last position. */
   window(3, 1, 14, 1);
   gotoxy(99,99);

   /* Get the current cursor position (should be 12, 1). */
   wherewincur(&x, &y);

   /* Display the last logical position in the current window. */
   printf("\fx=%d y=%d", x, y);
}
```

which_guard()

Returns the address of a trap handler.

Prototype

```
#include <trap.h>

proc = which_guard(can_handle_mask);

struct trap_mask can_handle_mask;
void   *proc;
```

Parameters `a_mask` is a trap mask that specifies the trap number.

Returns The function reference for the guard which yields the first non-zero result of the `can_handle_mask` ANDed with the guard's `take` mask. That is, the function which would actually handle the given trap if it were raised. If such an entry is found, `which_guard()` returns the associated function pointer. If no entry is found, NULL is returned.

Notes This function searches the system guard list (from most recent to oldest entry) for a guard yielding a non-zero result of the `a_mask` parameter and its `take` mask. The trap handler's address is returned for the first entry meeting this criteria. If no entry is found, a null pointer is returned.

Example See following page.

Example

```
/* WHICHGRD.C */

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

void handler()
{
  return;
}

main()
{
  struct trap_mask take, hide;

  void (* trap_proc)();

  /* SET UP TRAP MASKS */
  take=mask_of(1);
  hide=mask_of(1);

  /* SET HANDLER FUNCTION FOR TRAP 1 */
  trap_proc=handler;
  arm_guard (trap_proc, take, hide);

  /* FIND HANDLER FUNCTION FOR TRAP 1 */
  trap_proc = which_guard(take);
}
```

window()

Defines a logical window within the physical display.

Prototype

```
#include <txostd.h>
void window( x1, y1, x2, y2 );
int x1;
int y1;
int x2;
int y2;
```

Parameters

x1	Upper left column coordinate.
y1	Upper left row coordinate.
x2	Lower right column coordinate.
y2	Lower right row coordinate.

Returns VOID

Notes If any coordinates fall outside the physical display dimensions, the minimum/maximum values are used. The cursor is placed in the home position (1,1) of the window.

Segment-type display: The valid range for x-coordinates is 1 – 20; the valid range for y-coordinates is 1 or 2.

Pixel-type display: The maximum size for a window is determined by the current grid setting. See *Chapter 8, System Devices, Pixel-based Display*.

Pixel-type display: Special restrictions apply to this function when a double-wide font (e.g., Chinese font) is used: the window width must be an even number, the window must contain only double-wide characters, and the window cannot split any double-wide characters already on the display. These restrictions are not enforced by the firmware, but may result in undefined behavior if violated.

Related `clreol()`, `gotoxy()`

Example See following page.

Example

```
/* WINDOW.C */

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

int x1,y1,x2,y2;
main () {

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

   /* Define a window. */
   window(3, 1, 14, 1);

   /* Get window coordinates. */
   wherewin(&x1, &y1, &x2, &y2);

   /* Clear only the window and display window coordinates. */
   printf("\f(%d,%d) - (%d,%d)",x1, y1, x2, y2);
}
```

write()

Transfers data from the application's buffer to a file, or a device buffer.

Prototype

```
#include <io.h>

int write( handle, buffer, count );

int handle, count;
char *buffer;
```

Parameters

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

`buffer` is a pointer to a region of memory (not a counted string).

`count` is the number of bytes to be transferred.

Returns

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

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

Notes

When writing to a file, the current file position pointer determines where the data is written. After the write, the file position pointer is increased by the number of bytes actually written.

If the file position indicator is currently at the end of the file, the data written is appended to the file.

Example

See following page.

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);
}
```

write_cmd()

Sends modem commands.

Prototype

```
#include <device.h>
int  write_cmd( modem_handle, buffer, size );
int  modem_handle, size;
char *buffer;
```

Parameters

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

`size` is the maximum number of bytes to write.

`buffer` contains the data to be written.

Returns

`bytes_written` The number of bytes actually written.

Notes

The modem's response to a `write_cmd()` call should always be checked using `read_cmd()`.

Related

`read_cmd()`

Example

See following page.

Example

```
/* MODEMCMD.C */
#include <stdio.h>
#include <device.h>
int modem,bytes;
char buffer[10];
struct Opn_Blk opnblk;

main() {

   /* Open the modem. */
   modem = open("/dev/com4",0);

   /* Read past the two responses resulting from opening */
   /* the modem.  Each response should be 2 bytes long.  */
    for (bytes=0; bytes<4;)
       bytes += read_cmd(modem, &buffer[bytes], 2);

   /* Set up the open block and initialize using ioctl. */
   opnblk.rate=Rt_1200;
   opnblk.format=Fmt_A7E1;
   opnblk.protocol=P_pakt_mode;
   opnblk.parameter=0;
   opnblk.trailer.packet_parms.stx_char=2;
   opnblk.trailer.packet_parms.etx_char=3;
   opnblk.trailer.packet_parms.count=1;
   if (!ioctl(modem,0,&opnblk)) {
      printf("\fIOCTL SUCCESSFUL");
      getchar();
   }

   /* Write/Read a successful command (Set BELL Mode) */
   write_cmd(modem,"ATB1\015",5);
   while(!read_cmd(modem,buffer,10));
   printf("\fCOMMAND OK=%c",buffer[0]);
   getchar();

   /* Write/Read an erroneous command. */
   write_cmd(modem,"AT??\015",9);
   while(!read_cmd(modem,buffer,10));
   printf("\fCOMMAND ERROR=%c",buffer[0]);

   /* Close the modem. */
   close(modem);
}
```

write_cvlr()

Writes a single compressed variable-length record into a general-purpose file.

Prototype

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

Parameters

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

`buffer` contains the data to written.

`size` is the number of bytes to be written, starting at the current file position indicator.

Returns

`bytes_written`	The number of compressed bytes written.

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

Notes

If the file position indicator is positioned on an existing compressed-variable length record, that record is overwritten with the new data, as if a `delete_cvlr()` and `insert_cvlr()` has been performed. If it is positioned at the end of the file, the new record is appended to the file.

❖ *Warning: If the data to be written from the buffer contains ASCII bytes falling outside the range from 0x00 – 0x5F, they will not be translated correctly during the storage process.*

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 replace a record without mistranslating other data, the file position indicator must be positioned at the beginning of an existing compressed variable-length record.

Example

See following page.

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);
}
```

write_vlr()

Writes a single normal variable-length record into a general-purpose file.

Prototype

```
#include <vlr.h>

int write_vlr( handle, buffer, size );

int handle, size;
char *buffer;
```

Parameters

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

`buffer` contains the data to written.

`size` is the number of bytes to be written, starting at the current file position indicator.

Returns

`bytes_written`	The number of bytes written.
`-1`	Failure, with `errno` set to a specific error value. Values for `errno` include: EBADF EINVAL ENOSPC

Notes

If the file position indicator is positioned on an existing normal variable-length record, that record is overwritten with the new data, as if a `delete_vlr()` and `insert_vlr()` has been performed. If it is positioned at the end of the file, the new record is appended to the file.

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 replace a record without corrupting other data, the file position indicator must be positioned at the beginning of an existing variable-length record.

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", 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);
}
```

A P P E N D I X

A

International Certification Restrictions

This chapter provides information for developing applications that must meet certain country specific certification restrictions. This chapter does cover all restrictions for every country implementing OMNI 300 Series terminal applications.

❖ *Portions of this chapter have been written as a requirement for certification in specific countries. However, whenever developing an application that must meet the standards or restrictions of a specific country, all relevant standards and requirements for that country should be carefully examined and followed by the application programmer.*

U.K. BABT Restrictions

Important BABT information for U.K. users:

Improper use of certain Hayes-compatible modem commands may make the terminal non-compliant with one or more of the following BABT standards:

BS6305 : 1982
BS6789 : Part 1 : 1984
BS6789 : Section 3.1 : 1985
BS6789 : Section 3.2 : 1987

U.K. Users:

The following Hayes-compatible commands and register settings have restrictions that must be observed in a U.K. application:

- B Command
- &G Command
- &P Command
- X Command
- D Command, W parameter
- Hayes registers: S0, S6, S11, S53, S54, S55, S56, S63

Each restriction is described below:

B, &G, &P Hayes commands: None of these commands can be used. Such use will invalidate application certification.

X Hayes Command: Only modes X0, X1 and X2 are valid for application certification. Busy Tone Detect must be disabled.

D Hayes Command, W parameter: Use of the W (wait) parameter in a Dial string will invalidate approval to BS6789 : Section 3.1 : 1985. Dial Tone Detect must be enabled or disabled through the S60 Hayes register, only.

S0 Hayes register: Cannot be set to 1 or to a value greater than 5. Such setting will invalidate approval to BS6789 : Section 3.1 : 1987.

S6 Hayes register: Cannot be altered by the user in U.K. applications. Any attempt to set the S6 register results in an error message.

S11 Hayes register: Cannot be set to a value less than 70 (ms). Any S11 register value less than 70 results in an error message.

S53, S54, S55, S56, S63 Hayes registers: None can be altered by the user in U.K. applications. Any attempt to set the these register results in an error message.

A P P E N D I X

B

Error Return Codes

For most functions listed in Chapter 11, *Library Functions*, a return value of -1 is provided whenever an error condition is encountered. In addition to this general failure code, the value of the global variable `errno` is set to a specific error code indicating the general nature of the failure. These error codes, along with their symbolic names and meanings, are listed on the following pages in two forms, by error value and alphabetically by symbolic name.

Error Return Codes Listed by Value

Symbolic Name	Value	Definition
EPERM	1	No permission to access file
ENOENT	2	No such file or directory
ESRCH	3	No such process
EINTR	4	Interrupted system call
EIO	5	I/O error (physical)
ENXIO	6	No such device or address (or beyond limit)
E2BIG	7	Argument list too long: EXEC list > 5120 bytes
ENOEXEC	8	EXEC format error (file has no "magic" number)
EBADF	9	Bad file number (not open, write only, etc.)
ECHILD	10	No child processes (for WAIT)
EAGAIN	11	No more processes (FORK failure)
ENOMEM	12	Insufficient memory (EXEC or SBRK or FORK or...)
EACCES	13	Access denied by file protection system
EFAULT	14	Bad address: hardware fault using argument
ENOTBLK	15	Block device required: e.g., for MOUNT
EBUSY	16	Device busy (MOUNT if already mounted,....)