PIN Pad ioctl() Commands	
Command	**Command Description**
SetCtrl | 0	Port initialization
SetCtrl | 1	Reset error conditions
GetCtrl | 0	Retrieve port status
GetCtrl | 1	Get current Opn_Blk

SetCtrl | 0 — Port Initialization

The type parameter is set to SetCtrl | 0x0000, and buffer contains an Opn_Blk structure (defined in <device.h>). For Opn_Blk definitions, see Initializing Device Settings.

SetCtrl | 1 — Reset Error Conditions

Parity, framing, and overrun errors may be reset by setting the type parameter to 0x0001. A buffer address must be provided to satisfy the compiler, even though the buffer parameter is not actually used.

GetCtrl | 0 — Retrieve Port Status

When type is GetCtrl | 0, ioctl() returns 1 if pending output still exists, and 0 if no output is pending. In addition, this command provides current port status information.

This information is stored in a four-byte buffer as follows:

Byte 1: Number of input messages pending
Byte 2: Number of failed output messages pending
Byte 3: Number of output slots available
Byte 4: Current signal information:

0x80	Set if break/abort detected
0x40	Reserved, always zero
0x20	Set if CTS detected
0x10	Set if ring indicator present
0x08	Set if DCD present
0x04	Set if framing error detected
0x02	Set if overrun error detected
0x01	Set if parity error detected

Parity errors are latched until reset by calling ioctl() with SetCtrl | 1, as described earlier in this section.

The returned value in result will indicate whether or not any output is currently queued or being transmitted. This may be used to close the port while ensuring that all buffered data is sent.

GetCtrl | 1 **Get Current** Opn_Blk

Setting type to GetCtrl | 0x0001 copies the current Opn_Blk structure into the buffer. The buffer parameter should be passed as an address.

For Opn_Blk definitions, see Serial Port ioctl() SetCtrl | 0 description.

PIN Pad Example

```
/*********************************************************/
/* PINPAD.C                                              */
/* This program demonstrates the use of the PIN pad port */
/* functions. The program assumes a VeriFone PINpad 201  */
/* is connected to the PIN pad port.                     */
/*********************************************************/

#include <stdio.h>
#include <ioctl.h>
#include <device.h>
#include <config.h>
#include <ascii.h>
#include <io.h>
#include <memory.h>
#include <txosvc.h>

int pin_handle;
struct Opn_Blk pin_parm,ob_copy;
char buffer[100];
int sz;
```

```
main()
{
   /* Open the PIN pad. */
   pin_handle = open("/dev/pinpad", 0);

   /* PIN pad will communicate at 1200 baud, async, */
   /* 7 data bits, even parity, 1 stop bit,         */
   /* packet mode with STX, ETX, and LRC.           */
   pin_parm.rate=Rt_1200;
   pin_parm.format=Fmt_A7E1;
   pin_parm.protocol=P_pakt_mode;
   pin_parm.parameter=0;
   pin_parm.trailer.packet_parms.stx_char=STX;
   pin_parm.trailer.packet_parms.etx_char=ETX;
   pin_parm.trailer.packet_parms.count=1;
   ioctl(pin_handle, SetCtrl | 0, &pin_parm);

   /* Get a copy of the current open block. */
   /* Just here to show how it is done.     */
   ioctl(pin_handle, GetCtrl | 1, &ob_copy);

   /* Send the "Q1" command to the PINpad 201.  */
   /* The PIN pad will display "SLIDE CARD".    */
   buffer[0] = STX;
   memcpy(&buffer[1], "Q1", 2);
   buffer[3] = ETX;
   buffer[4] = SVC_CRC_CALC(0, &buffer[1], 3);
   write(pin_handle, buffer, 5);

   /* Get the response from the PINpad 201;  */
   /* should be an ACK.                      */
   while (read(pin_handle, buffer, 1) == 0);
   if (buffer[0] == ACK) printf("\fPINPAD READY");

   /* Wait for card to be swiped on PIN pad.          */
   /* Packet returned: <STX>81.<card data><ETX><lrc> */
   while ((sz=read(pin_handle,buffer,sizeof(buffer)))==0);

   /* If LRC is valid, display card data. */
   if (buffer[sz-1] == SVC_CRC_CALC(0,&buffer[1],sz-2))
      write(STDOUT, &buffer[4], 16);
```

```
    /* Send an ACK, then wait for it to make it out  */
    /* of the PIN pad port before we close the port. */
    /* Otherwise the ACK may be lost.                */
    write (pin_handle, "\006", 1);
    while (ioctl(pin_handle, GetCtrl | 0, buffer));
    close(pin_handle);

    /* Display the buffer received from the GetCtrl | 0 */
    /* above.                                           */
    SVC_WAIT(1500); printf("\f%d INPUT MSGS PND",buffer[0]);
    SVC_WAIT(1500); printf("\f%d FAILED OUTPUTS",buffer[1]);
    SVC_WAIT(1500); printf("\f%d SLOTS AVAIL",   buffer[2]);
    SVC_WAIT(1500); printf("\fSIGNAL INFO: %x",  buffer[3]);
}
```

Modem Interface

380 DIAL
385 DIAL
390 DIAL
395 DIAL
460 DIAL

The information in this section describes the modem interface of the internal modem and COM1 ports of dial-type models, and the COM1 and COM3 ports of the LAN-type models. Variations and limitations of particular ports/models are noted in the text.

The modem interface has been designed to take advantage of industry modem standards, as well as allow for non-standard or international modem applications. Both internal and external modem communications support Hayes-compatible modem functionality when possible. However, unlike Hayes modems, OMNI 300 Series terminals use separate command and data paths, allowing an application to send both data and commands simultaneously. The `read_cmd()` and `write_cmd()` modem functions, which use this feature, are available only to the internal modem.

❖ *Some modem commands or Hayes-type register settings are disallowed or are restricted by country-specific application/firmware certification rules. Refer to Appendix A, International Restrictions, for country-specific restrictions.*

Modem Async/Sync Modes

OMNI 300 Series terminals can communicate in either asynchronous or synchronous mode. The following modes of asynchronous communications are available:

Character
Packet
VISA First Generation
ZONTALK Downloading (via SVC_ZONTALK())

Synchronous communication modes are:

SDLC
SDLC Fast Connect (for compatibility with Hypercom®)

SDLC Support

The internal modem of dial-type terminals includes support for SDLC communications. The COM3 port of LAN-type terminals will support SDLC if connected to an appropriate external modem, but will not support DTR signals. For a definitive review of SDLC, see *Synchronous Data Link Control Concepts*, IBM document GA27-3093-3, 4th edition (June 1986).

OMNI 300 Series terminals support a subset of the packet formats defined for secondary stations in a half-duplex switched point-to-point configuration. IBM has defined three major groups of packet formats: Supervisory, Information, and Unnumbered.

Supervisor Format

OMNI 300 Series terminals can receive all supervisory format frames. Only "ready-to-receive" packets are processed or can be generated.

Information Format

OMNI 300 Series terminals can generate and receive Modulo-8 information packets. Each packet received is immediately acknowledged; each packet sent should be acknowledged immediately, as there is no provision for more than one unacknowledged packet. Modulo-128 packets are not supported.

Unnumbered Format

All unnumbered information packet formats can be received. The "set normal response mode" packet is processed and

acknowledged with the "unnumbered acknowledgment" packet. The "test" packet is echoed back to the sender. The "disconnect" message is acknowledged with the "unnumbered acknowledgment" packet, but will not otherwise be processed. All other unnumbered information packets are ignored.

The actual communication begins as soon as the modem handshake is complete. The handshake can be a normal Bell 212A handshake or a special "fast connect" version used by Hypercom Network Access Controllers. Once the handshake is complete, the terminal will continuously generate SDLC "flag" characters, except when transmitting SDLC frames.

Modem Functions

open()

Prepares the modem for subsequent use.

```
#include <io.h>

hMODEM = open("/dev/com4", 0);
int hMODEM;
```

The modem remains inactive until a call to `ioctl()` is made to set the baud rate, format and protocol parameters. When the modem is opened, all of the configurable S-registers (discussed later) are reset to their default values in a two-step process. The first step restores the "factory defaults." The second step attempts to set the S-registers to the "user default values" as found in the CONFIG.SYS *MI Parameter. This step may fail if the user's settings are incorrect.

Constant	Baud Rate
Rt_300	300
Rt_600	600
Rt_1200	1200
Rt_2400	2400
Rt_4800	4800
Rt_9600	9600
Rt_19200	19,200

Presently, only 300 and 1200 are viable baud rates for OMNI 300 Series terminals.

read()

Transfers data from the modem to the application's buffer.

```
#include <io.h>

bytes_read = read (hMODEM, buffer, size);
int  bytes_read, hMODEM, size;
char *buffer;
```

Each `read()` transfers `size` bytes from the communications (modem) buffer into the application buffer. It returns the number of bytes actually read: 0 if no data is available, or -1 if an error occurs.

Unread buffered data is handled differently, depending on the mode. In packet mode, up to `size` bytes are moved into `buffer`; any portion of the packet larger than `size` bytes is truncated and lost. In character mode, up to `size` bytes are moved into the `buffer`; if more data is available, it remains in the communications buffer for the next `read()` operation.

write()

Transfers data from the application's buffer to the modem.

```
#include <io.h>

bytes_written = write (hMODEM, buffer, size);
int bytes_written, hMODEM, size;
char *buffer;
```

Each `write()` transfers `size` bytes from `buffer` to the modem's buffer. If the operation is successful, `bytes_written` equals `size`. If the modem has no available buffer space or an error occurs, -1 is returned.

Once in the modem buffer, the data is transferred to the transmitter each time the transmit buffer goes empty.

read_cmd()

Reads a command response from the modem.

```
#include <device.h>

bytes_read = read_cmd(HMODEM, buffer, size);
int bytes_read, hMODEM, size;
char *buffer
```

This function transfers a maximum of `size` bytes into `buffer`; if more command response data is available, the data will remain in the modem's buffer for the next `read_cmd()` operation. `bytes_read` returns the number of bytes actually read, or 0 if no command response data is available.

Typically, `read_cmd()` should be called after every `write_cmd()` to check for a response. In addition, `read_cmd()` should be called to get the two command responses following an `open()` (see Examples). Be aware that responses may occur substantially later than the call—the response to a dial command may occur many seconds after the command was issued.

write_cmd()

Sends commands to the modem.

```
#include <device.h>

bytes_written = write_cmd(hMODEM, buffer, size);
int hMODEM, size;
char *buffer;
```

This function transfers `size` bytes of command data from `buffer` to the modem. `bytes_written` should equal `size`. The modem's response to a `write_cmd()` should always be checked using `read_cmd()`.

close()

Releases the resources associated with the modem.

```
#include <io.h>

status = close(hMODEM);
int status, hMODEM
```

If the operation succeeds, `status` is set to zero. If an error occurs, it is set to -1.

ioctl()

Several modem operations may be performed using `ioctl()`, including:

- Modem initialization
- Resetting error conditions
- Resetting BRK, RTS and DTR control signals
- Retrieving reject queue
- Retrieving device status

The function prototype is:

```
#include <io.h>

result = ioctl (hMODEM, type,buffer);
int  result,hMODEM;
unsigned int type;
void *buffer;
```

The function normally returns 0 if the operation succeeds, or -1 if an error occurs. One exception is the `GetCtrl|0` command, which returns a 1 if output is pending for the device.

As with other `ioctl()` operations, the `type` parameter determines the operation to be performed. Each command is described briefly in the following table, and in detail on the following pages.

Modem `ioctl()` Commands

Command	Command Description
`SetCtrl|0`	Modem initialization
`SetCtrl|1`	Reset error conditions
`SetCtrl|2`	Start/stop break signal (BRK)
`SetCtrl|3`	Send a command
`GetCtrl|0`	Checks pending output, reads status bytes
`GetCtrl|1`	Get `Opn_Blk` structure
`GetCtrl|2`	Read reject queue
`GetCtrl|3`	Read a command

`SetCtrl | 0` — Modem Initialization

To initialize the modem, set the type parameter to `SetCtrl | 0x0000`, and the `buffer` parameter to an `Opn_Blk` structure (defined in `<device.h>`). This structure controls port baud rate, data format, protocol and other parameters for subsequent modem operations.)See *Initializing Device Settings*.)

❖ *This operation must be done before calling any read or write routines.*

SetCtrl | 1 — Reset Error Conditions

Resets error conditions for parity, framing and overrun. A buffer address must be provided to satisfy the compiler, although the buffer parameter is not actually used.

SetCtrl | 2 — Start/Stop Break Signal

To turn the break signal on, use the command:

```
ioctl (hMODEM, SetCtl | 2, &buffer);
```

where buffer[0]=4

To turn the break signal off, use the command:

```
ioctl (hMODEM, SetCtl | 2, &buffer);
```

where buffer[0]=0

SetCtrl | 3 — Send a Command

`buffer` should be a structure of type:

```
struct {char *buf; int size;} buffer;
```

where `buf` is a pointer to the data area containing the modem command string, and `size` is the size of the string. `buffer` should be passed as `&buffer`.

This function serves the same purpose as `write_cmd()`, and is included for backward compatibility only. `write_cmd()` should be used.

GetCtrl | 0 — Check Modem Status

In character or packet mode, `result` is set to 1 if data is waiting to go out of the modem port, or 0 if no output is pending. Call this command prior to closing the modem to avoid truncating the data transfer. When communicating in other than character or packet mode, `result` is not meaningful.

Four bytes of status information are copied to the buffer:

Byte 1: Number of input messages pending.

Byte 2: Number of failed output messages pending.

Byte 3: Number of output slots available—computed as maximum slots less slots in use. Be aware that there may not be enough buffers available for all the slots required.

Byte 4: Current signal information:

0x80	Set if break/abort detected
0x40	Reserved, always zero
0x20	Set if CTS detected
0x10	Set if ring indicator present
0x08	Set if DCD present
0x04	Set if framing error detected
0x02	Set if overrun error detected
0x01	Set if parity error detected

Parity errors are latched until reset by calling `ioctl()` with `SetCtrl | 1`, as described earlier in this subsection.

The following function may be used to close the port while ensuring that all buffered data is sent:

```
while  (ioctl(hCOM, GetCtrl | 0, &buffer) != 0)
          printf ("\fOUTPUT PENDING");
close (hCOM);
```

`GetCtrl | 1` — Get Opn_Blk Structure

Copies the current `Opn_Blk` structure into the `buffer`; `buffer` should be passed as `&buffer`.

`GetCtrl | 2` — Read Reject Queue

Reads and removes a record from the reject queue and puts it in `reject_message`. `buffer` should be a structure of the type:

```
struct { char *buf; int size; } buffer;
char reject_message[200];
buffer.buf = reject_message;
buffer.size = sizeof(reject_message);
ioctl(mdm_handle, GetCtrl | 2, &buffer);
```

In this definition, `buf` is a pointer to `reject_message` and `size` is the maximum size of `reject_message`.

`buffer` should be passed as `&buffer`.

`P_visalgen` is the only protocol that will add records to the reject queue. When an output record, created via `write()`, cannot be delivered to the remote communication device, the record, and all other pending records, are added to the reject queue. Whenever a record is added to the queue, Trap 48 is triggered (see *Chapter 7, Exception Handling*).

- *Each* `P_visalgen` *record created will occupy a buffer from the common buffer pool. The only way to re-use this buffer is to read the record from the reject queue (or re-open the port).*

`GetCtrl | 3` — Read a Command

Read a command response from the modem. `buffer` should be a structure of type:

```
struct { char *buf; int size; } buffer;
```

where `buf` is a pointer to a data area which will receive the command response string and `size` is the maximum size of the data area. `buffer` should be passed as `&buffer`. This function serves the same purpose as `read_cmd()`, and is included for backward compatibility only. `read_cmd()` should be used.

Modem Host Communications Example

```
/*************************************************************/
/* MDMHOST.C                                                 */
/* This program demonstrates the use of the MODEM device     */
/* by communicating with another 380.  This program operates */
/* in character mode and communicates to MDMVISA.C which     */
/* functions in VISA mode.                                   */
/*                                                           */
/* Download the programs to separate OMNI 380's and start    */
/* MDMHOST first, then MDMVISA. The communication between    */
/* the two will be:                                          */
/*                                                           */
/*   MDMHOST                    MDMVISA                      */
/*   =======                    =======                      */
/*   <ENQ>                                                   */
/*                              <STX>REQUEST<ETX><lrc>       */
/*   <ACK>                                                   */
/*   <STX>APPROVED<ETX><lrc>                                 */
/*                              <ACK>                        */
/*   <EOT>                                                   */
/*                                                           */
/*************************************************************/

#include <stdio.h>
#include <device.h>
#include <io.h>
#include <ascii.h>
#include <strings.h>
#include <memory.h>
#include <txosvc.h>

int mdm_handle,bytes,i;
char buffer[20];
struct Opn_Blk mdm_Opn_Blk;

main () {

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

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

```
/* Define the protocol as:
      1200 baud, async, 7 data bits, even parity,
      and character mode.
*/
mdm_Opn_Blk.rate = Rt_1200;
mdm_Opn_Blk.format = Fmt_A7E1;
mdm_Opn_Blk.protocol = P_char_mode;
ioctl(mdm_handle, SetCtrl | 0, &mdm_Opn_Blk);

/* Enable answer mode by setting S0=1 */
write_cmd(mdm_handle, "ATS0=1\r", 7);
while(!(bytes=read_cmd(mdm_handle,buffer,10)));
if (buffer[0] == '0') printf("\fWAITING FOR CALL");

/* Wait for a successful connection. */
do
   bytes=read_cmd(mdm_handle,buffer,10);
while (buffer[bytes-2] != '5');
printf("\fCONNECTED");

/* Send an ENQ to initiate a transaction. */
buffer[0]=ENQ;
write(mdm_handle, buffer, 1);

/* Read the first 10 bytes received.  */
/* Expecting <STX>REQUEST<ETX><lrc>   */
for (i=0; i<10; i++)
   while (read(mdm_handle, &buffer[i], 1) == 0);

/* Check for LRC error. */
if (buffer[9] != SVC_CRC_CALC(0, &buffer[1], 8))
   printf("\fLRC ERROR!");

/* Send an ACK to indicate the request was received. */
buffer[0]=ACK;
write(mdm_handle, buffer, 1);

/* Create a buffer with STX, "APPROVED", ETX and LRC. */
buffer[0]=STX;
strcpy(&buffer[1],"APPROVED");
buffer[9]=ETX;
buffer[10]=SVC_CRC_CALC(0, &buffer[1], 9);
```

```
    /* Output the buffer. */
    write(mdm_handle, buffer, 11);

    /* Wait for an ACK */
    while (read(mdm_handle, buffer, 1) == 0);

    /* If ACK, send EOT to end the transaction. */
    if (buffer[0] == ACK) {
       printf("\fACK RECEIVED");
       buffer[0]=EOT;
       while (write(mdm_handle, buffer, 1) != 1);
    }

    /* Wait for all the output to clear out before */
    /* we close the port or data may be lost.      */
    while (ioctl(mdm_handle, GetCtrl | 0, buffer));
    close(mdm_handle);

    /* Display the buffer received from the GetCtrl | 0 */
    /* above.                                           */
    SVC_WAIT(1500); printf("\f%d INPUT MSGS PND",buffer[0]);
    SVC_WAIT(1500); printf("\f%d FAILED OUTPUTS",buffer[1]);
    SVC_WAIT(1500); printf("\f%d SLOTS AVAIL",   buffer[2]);
    SVC_WAIT(1500); printf("\fSIGNAL INFO: %x",  buffer[3]);
}
```

Modem VISA Example

```
/*************************************************************/
/* MDMVISA.C                                                 */
/* This program demonstrates the use of the MODEM device     */
/* by communicating with another 380.  This program operates */
/* in VISA mode and communicates to MDMHOST.C which          */
/* functions in character mode.                              */
/*                                                           */
/* Download the programs to separate OMNI 380's and start    */
/* MDMHOST first, then MDMVISA.  The communication between   */
/* the two will be:                                          */
/*                                                           */
/*   MDMHOST                    MDMVISA                      */
/*   =======                    =======                      */
/*   <ENQ>                                                   */
/*                              <STX>REQUEST<ETX><lrc>       */
/*   <ACK>                                                   */
/*   <STX>APPROVED<ETX><lrc>                                 */
/*                              <ACK>                        */
/*   <EOT>                                                   */
/*                                                           */
/*************************************************************/


#include <stdio.h>
#include <config.h>
#include <device.h>
#include <io.h>
#include <ascii.h>
#include <strings.h>

int mdm_handle,bytes_read,i,bytes;
char buffer[20];
struct Opn_Blk mdm_Opn_Blk,new_Opn_Blk;

main () {

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

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

```
    /* Define the protocol as:
          1200 baud, async, 7 data bits, even parity,
          and VISA mode.
    */
    mdm_Opn_Blk.rate = Rt_1200;
    mdm_Opn_Blk.format = Fmt_A7E1;
    mdm_Opn_Blk.protocol = P_visalgen;
    ioctl(mdm_handle, SetCtrl | 0, &mdm_Opn_Blk);

    /* Get a copy of the open block and modify VISA
       mode to send one <ACK> sent after the approval.
       For demonstration purposes only, normally
       include this initialization above.
    */
    ioctl(mdm_handle, GetCtrl | 1, &new_Opn_Blk);
    new_Opn_Blk.trailer.other_parms.other_parm[0] = 0x21;
    ioctl(mdm_handle, SetCtrl | 0, &new_Opn_Blk);

    /* Call the number MDMHOST is attached to. */
    printf("\fCALLING HOST");
    write_cmd(mdm_handle, "ATD3392\r", 8);

    /* Wait for a connection. */
    do
       bytes=read_cmd(mdm_handle,buffer,10);
    while (buffer[bytes-2] != '5');
    printf("\fCONNECTED");

    /* Let VISA mode output <STX>REQUEST<ETX><lrc> */
    printf("\fSENDING REQUEST");
    write(mdm_handle, "REQUEST", 7);

    /* Wait for a response of APPROVED */
    while (read(mdm_handle, buffer, 20) == 0);
    if (!strncmp(buffer, "APPROVED",8))
       printf("\fAPPROVAL RECV'D");

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

Modem Command 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);
}
```

Hayes-Compatible Modem Control

❖ *Some modem commands or Hayes-type register settings are disallowed or are restricted by country-specific application/ firmware certification rules. Refer to Appendix A. International Restrictions for country-specific application implementation information.*

This section describes the Hayes-compatible modem commands and responses supported by the OMNI 300 Series terminals. In addition, this section lists modem commands required in OMNI 300 Series terminals that are not supported by Hayes commands.

OMNI 300 Series terminals send commands to the modem in a significantly different manner than the Hayes method. While Hayes-compatible commands and data are sent to the same place, OMNI 300 Series terminals use separate command and data paths.

Because of this, both data and commands may be sent at the same time.

- Data is sent to the modem using `write()` and is received from the modem using `read()`.
- Commands are sent to the modem using the `write_cmd()` function. Command response codes are received using the `read_cmd()` function.

Sending Commands

Once the modem has been opened, modem commands are sent to it using the `write_cmd()` function.

Hayes-compatible command strings normally start with the characters "AT" and terminate with a "<CR>" character, 0x0D (standard C escape code "\r" can be used). The command string may contain up to 40 characters (maximum), excluding the "AT", as in the example:

```
"ATD555-7825\r"
```

Each command (delimited by "AT" and "<CR>") is completely parsed and validated prior to execution. If any syntactical problems are detected, the entire command is rejected and the appropriate response code should be read. The `write_cmd()` function always returns the number of bytes written, even if the command is rejected.

Applications should always read the modem response using `read_cmd()` after sending a command.

❖ *If the modem is in the process of waiting for a connection to the remote host, issuing any new command terminates the call.*

Receiving Responses

The modem responses are received using the `read_cmd()` function. Responses are terminated by a carriage return character, "<CR>". The responses are received as ASCII characters values, hence the response code "90" consists of 2 characters, 0x39 and 0x30, not the value 90.

The valid response codes are:

Response Code	Code Description	See Note
"0"	Command completed	1
"1"	Connection made at 300 baud	1
"2"	Ring signal detected	1
"3"	No carrier/connection failed	1
"4"	Command error	1
"5"	Connection made at 1200 baud	1
"6"	Requested dialtone not detected	1
"7"	Requested busytone detected	1
"10"	Connection made at 2400 baud	1
"90"	Response code for NO LINE	2
"91"	Response code for DIALING COMPLETE	3

Notes

[1] Standard Hayes command response.

[2] VeriFone enhancement. Allows platform to determine whether line is connected before dialing or checking for dial tone.

[3] VeriFone enhancement. Allows terminal to determine when phone number dialing is complete.

If a command is not supported, or is disallowed due to a country-specific restriction, the modem returns code "4", command error.

Hayes-compatible Commands

Valid Hayes-compatible modem commands are sent to the modem with the `write_cmd()` function. Command strings are not case sensitive.

Valid Hayes commands while not dialing (on-hook) are listed in the table below and described on the following pages.

Hex Code	ASCII Code	Command Description	See Note
0x2D	-	Check line ready	1
0x44	D	Go off-hook, begin dialing	2
0x48	H	Hangup	
0x4F	O	Go into data mode	
0x53	S	Set current S-registers	
0x58h	X	Enable/disable call progress	
0x42	B	Select Bell/CCITT mode	3
0x26/0x47	&G	Toggle CCITT guard tones	3
0x26h/0x50	&P	Select pulse make-break ratio	3

Notes

Any invalid command character causes the modem to return response code "4", error. An invalid parameter used in a valid command also generates response code "4".

[1] VeriFone extension.

[2] After receiving a `D` command, the modem completely ignores any further on-hook commands. If an invalid command or parameter is issued, the modem does not return "4" to signal an error.

[3] International terminal versions supporting CCITT only.

Character and Command Rules

The space character is always ignored, since spaces are frequently inserted into the dialing string to enhance readability. In addition, the hyphen "-" and parentheses "()" are also ignored in a dial string.

Some unimplemented commands are always ignored, others are ignored only when they occur with a specified parameter value, and only if they are attempting to set values already in effect by default.

❖ *Do not under any circumstances use a TAB character (ASCII 009) in a command, as it is certain to cause an error.*

(-) Check Line Presence

The hyphen (-) command immediately checks for the presence of a telephone line. If no telephone line is present, the modem returns the response code "90", no line. Keep in mind that this code is two ASCII digits (0x39 0x30), followed (as always) by a carriage return.

This command does not force dialing or force the modem into on-line mode.

Bn Set Bell/CCITT Mode

The B command selects Bell or CCITT standard carrier frequencies and connection protocols (International CCITT terminals only).

- B0 selects the CCITT standard
- B1 selects the Bell standard for a given baud rate.

The default is country-dependent. In U.S. terminal versions, "1", Bell standard, is the default. Any value other than 0 or 1 results in an error.

Dnnn-nnnn Go Off-hook, Dial

The D command checks for the presence of a telephone line and, if present, dials its number. If no telephone line is present, the modem returns response code "90", no line.

If the line is present (or line testing is disabled), the modem dials the remaining characters in the command string (i.e., the telephone number). After the dial string is sent, the modem waits for a carrier, and goes on-line if it is detected.

H — Disconnect or Hang up

The H command instructs the unit to go on-hook, or disconnect from the telephone line. If the unit is already disconnected (on-hook), the command is ignored but still returns a "0", OK, response.

Note that 'ATH1' is not a valid command.

O — Go Back On-line

The Hayes O command (letter O, not zero) instructs the unit to go back into data mode from command mode. If the modem has not established a telephone line connection, an error response is returned. Since OMNI 300 series terminals support separate data and command modes, this command only checks whether or not the unit is connected.

Sr=nnn/Sr? — Set/Read S-register Values

The S command sets register r to the value nnn, allowing the user to alter certain modem settings and timings. The S-registers are initially set from the CONFIG.SYS file *MI entry when the device is opened. Some valid register values may be restricted by EPROM flags per international requirements. If the value specified by the S command falls outside the minimum or maximum value, the register's current value is returned and the modem reports an error. Valid S-registers are listed later under Setting the *MI Parameter

The value of each supported S-register can be read using the syntax Sr?. The result is returned as a three-digit decimal number followed by a carriage return. Example:

```
write_cmd(modem, "AT S0=5 S0? S57=15 S57? \r", 25);
SVC_WAIT(100);
read_cmd(modem, buffer, sizeof(buffer));
/* buffer now contains "005\r015\r0\r" */
```

Xn — Enable Call Progress

The X command enables or disables call progress capabilities. By default, Dial Tone Detect and Busy Tone Detect are enabled (X4). Default settings are retained until explicitly changed by another command, or when the device is closed or reopened.

Valid parameters are:

	Dial Tone Detect	Busy Tone Detect	Result Codes
X0	disabled	disabled	0 – 5, 10, 90
X1	disabled	disabled	0 – 5, 10, 90
X2	enabled	disabled	0 – 6, 10, 90
X3	disabled	enabled	0 – 5, 7, 10, 90
X4	enabled	enabled	0 – 7, 10, 90

Default: X4

The X0 command differs from its corresponding Hayes command in that X0 does not force the modem into 300 baud mode.

Dial Tone Detect

When Dial Tone detection is enabled, the unit goes off-hook and waits five seconds for a dial tone. If no dial tone is detected within the five seconds, the modem returns response code "6". If the unit detects dial tone, it processes the D command.

When Dial Tone Detect is disabled, the modem waits for two seconds, or a time specified in the S6 register, before dialing.

Busy Tone Detect

When Busy Tone Detect is enabled, the modem looks for a busy tone. The modem also looks for a busy tone when executing a W command (wait for secondary dial tone) or when it has finished dialing all characters and is looking for a carrier.

If the unit detects a busy tone, it returns response code "7".

When Busy Tone Detect is disabled, the unit only waits for and responds to a carrier.

&G Enable CCITT Guard Tones

If the terminal supports CCITT V.22 answer mode, guard tones may be enabled or disabled using the &G command. Value may range from 0-2.

- &G0 forces the modem to use no guard tone
- &G1 forces the modem to use a 550 Hz guard tone
- &G2 forces the modem to use an 1800 Hz guard tone

The default is &G0 (no guard tone).

This command is equivalent to setting the S63 register and is available only in international version terminals.

&P Set Pulse Dial Ratio

The &P command selects make/break ratios when pulse dialing. It assumes that the connection has not yet been made and that the unit is not dialing or waiting for carrier.

- &P0 selects 40/60 make-break ratio
- &P1 selects 33/67 make-break ratio.

The default is country-dependent—permanently set to 0 on USA terminal versions.

Make and break times can also be set explicitly using S54 and S55. This command is available only in international version terminals.

Off-Hook and Dialing Commands

Once the unit goes off-hook to dial (via the D command), the valid command characters are referred to as dial modifiers; they are:

Hex Code	ASCII Code	Command Description
0x30-0x39	0-9	Digits, pulse or tone dial
0x41-0x44	A-D	Digits, tone dial only
0x23, 0x2A	# *	Digits, tone dial only
0x50	P	Switch to pulse dial
0x54	T	Switch to tone dial
0x56	V	Verify answer tone (Bell 103/Bell 212)
0x57	W	Wait for dial tone, or (S7) secs
0x2C	,	Wait for (S8) seconds
0x3B	;	Eliminate the handshake
Characters ignored in a dial modifier:		
0x20	" "	Space character
0x2D	-	Hyphen character
0x28	(	Left parenthesis
0x29	)	Right parenthesis

The use of any other character causes a return code of "4", error. Any unimplemented digits or dialing modifiers also cause a return code "4".

0-9, A-D, # * — Dialing Digits

Dialing digits are characters accepted by a telephone service to dial a call. The digits "A-D", "#", and "*" are only valid if tone dialing. Each digit is dialed as it is encountered in the string.

, — Pause Character

The comma (,) command causes the unit to delay a set period of time before proceeding to execute the remaining commands. This delay may be changed via the S8 register.

; — Eliminate Handshake

To eliminate modem handshaking, terminate the dial string with a semicolon (;). This feature allows applications to perform speed dialing for voice operation.

P/T — Pulse/Tone

OMNI 300 Series terminals are able to dial in either pulse or tone. The commands P (pulse) or T (tone) select the dialing method to be used in subsequent dialing.

The default is tone. This setting is retained until explicitly changed by another T or P command, or the device is closed or re-opened. The duration of the tone is set by the S11 register.

V — Verify Answer Tone

In the Bell mode, the modem does not require the remote host to generate the appropriate answer tone as part of the standard connect sequence. Setting the V option forces the modem to require an answer tone as part of the connect sequence. This command only affects the current connection; it is reset to "no verification" following the call.

W — Wait For Secondary Dial Tone

The W command instructs the modem to wait for a dial tone, or until a 30-second period elapses, before proceeding to execute the remaining commands. If the unit does not detect a dial tone within this time period, it hangs up, returning response code "6". The wait period (default 30 seconds) may be changed by setting the modem S7 register.

Modem EPROM Initialization

The modem device driver uses an EPROM table to configure the modem/telco circuitry and software. This table contains an entry for each parameter that can be altered by the user or the application.

When the modem device is opened, all default values stored in the EPROM table are set. This corresponds to "factory initialization" and will always return a "0", OK, result code (use the `read_cmd()` function to check). Following EPROM initialization, the system checks the *MI entry in CONFIG.SYS (if present). If the CONFIG.SYS entries are valid, the modem sends another "0" response or a "4" if any error occurs. Reading the *MI entry is a single command; hence, it must be entirely correct in order to be processed.

Modem CONFIG.SYS Initialization

Use the CONFIG.SYS file to change modem default settings stored in EPROM. Those values not initialized in CONFIG.SYS are listed as <EMPTY> in the terminal's Store/Recall mode.

❖ *Various countries prohibit the altering of EPROM entries. See Appendix A for more details.*

Setting the *MI Parameter

User modem initialization parameters are set as a record in the CONFIG.SYS file at the identifier *MI. This record is optional, but allows the user to set any of the S-registers. The format should include the correct command(s) but without the "AT" prefix and "<CR>" suffix. Spaces in the record are optional. An example of a *MI entry is:

```
*MI= S0=2 S59=1 S54=20 S55=30
```

This example enables auto-answer on the second ring (S0=2) and selects pulse dialing at 20 pulses per second (S59=1) with a 40/60 make/break ratio (S54=20 S55=30). Since this setting uses the S54 and S55 registers, it is valid only on international version terminals.

Hayes-compatible Modem Registers

The following Hayes-type modem registers can also be set in CONFIG.SYS. To set register Sn1 to a value v1, insert the following line into the CONFIG.SYS file:

```
*MI= Sn1=v1 Sn2=v2 Sn3=v3 ...
```

The application may also set any of these modem registers executing a command string containing the sequence Sr=nnn, for example:

```
write_cmd(hMODEM, "ATS0=1\r", 7);
```

An S-register's value can be retrieved by writing the command "Snnn?". The result is a three-digit decimal number and a carriage return; for example:

```
/* Write S-register values, with queries      */
write_cmd(hMODEM,"AT S0=5 S0? S57=15 S57?\r",13);
SVC_WAIT(100);
read_cmd(hMODEM, buffer, sizeof(buffer));
/* Buffer now contains: "005\r015\r0\r"       */
/* S0=5 -> 005\r                              */
/* S57=15 -> 015\r                            */
```

The table below lists valid user-configurable S-registers. The Range column shows the greatest possible range of settings (not firmware dependent). Terminal firmware may not allow all possible values. This table lists the highest possible setting on U.S. terminals, as well as the default settings for U.S. terminal firmware. Other maximum and default values depend upon the firmware released for a specific country.

Register	Range	U.S. Max.	U.S. Default	Description
S0	0–225 rings	25	0	*Auto-answer rings (0=Disable)*
S6	0–255 sec.	255	2	*Wait time before blind dialing*
S7	1–55 sec.	55	30	*Wait time for carrier/dial tone*
S8	0 –255 sec.	255	2	*Pause time*
S9	1–255 0.1 sec.	255	6	*Carrier tone validation*
S11	25–255 ms.	255	60	*DTMF tone duration*
S50	0–3, flag	0	0	*Select relay use*
S51	5–250 ms.	250	10	*Auxiliary Relay switch in time*
S52	5–250 ms.	250	10	*Auxiliary Relay switch out time*
S53	0–1, flags	1	1	*Modem type (0 = Bell)*
S54	20 –100 ms.	100	40	*Pulse-dialing "make" time*
S55	20 –100 ms.	100	60	*Pulse-dialing "break" time*
S56	8–30 50 ms.	30	16	*Pulse-dialing "gap" time*
S57	10–83 Hertz	83	15	*Ring cycle, min. frequency*
S58	10–83 Hertz	83	66	*Ring cycle, max. frequency*
S59	0–3, flag	1	0	*Dial type (0 = Tone)*
S60	0–1, flag	1	0	*Dial tone check (0 = Enable)*
S61	0–1, flag	1	0	*Busy signal check (0 = Enable)*
S62	0–1, flag	1	0	*Line check (0 = Enable)*
S63	0–2, flag	1	0	*Guard Tone (CCITT)*
S64	0-15 samples/ .25 second	15	8	*Threshold value for DCD loss*
S65	0–1, flag	0	0	*V.21 Answer tone checking*
S66	0–1, flags	0	0	*V.22 Unscrambled marks checking*
S67	0–1, flags	1	1	*Dialing complete (result code "91") check (0 = Enable)*

S0 Rings Before Answering

The S0 register contains the number of rings the terminal must see before answering. If 0 is entered, auto answer is disabled.

S6 Wait Time Before Blind Dialing

The value in the S6 register determines how long the modem waits after going off hook before it dials. This delay allows time for the central telephone office to detect the off-hook condition of the line.

If dial tone detection is enabled, blind dialing is disabled and this register is irrelevant.

S7 Wait Time For Carrier/Dial Tone

The S7 register serves two functions. First, when establishing a call, the value in S7 determines the modem's time delay between dialing and responding to an incoming carrier signal after initial connection; if the terminal does not detect carrier within this time, the terminal hangs up and returns response code "3", no carrier. If the terminal detects carrier within the specified time, it goes on line.

The S7 register also determines the wait duration for a secondary dial tone (generated by the W dial modifier).

- Valid range is 1–55 seconds, in 1-second increments
- U.S. default: 30 seconds
- Initial dial tone hard-coded to 5 seconds

S8 Pause Time

The S8 register determines the duration of a pause generated by the comma (,) dial character.

Its range is 0 to 255 seconds in increments of 1 second. The U.S. default is 2 seconds.

S9 Carrier Tone Validation

The S9 value sets the period of time the modem carrier tone must be present before it can be regarded as valid and reported by the modem driver.

S11 DTMF Tone Duration

The S11 value determines the duration and spacing of tones in DTMF dialing. It does not affect the DTMF duty cycle, which is 50/50 on-off by default (U.S.). This value has no effect on the pulse dialing speed, which is fixed at 10 pulses per second.

S50 Select Relay Use

OMNI 300 Series terminals have an auxiliary relay that, depending upon terminal hardware (international versions), controls various modem features such as A/A1 support, spark suppression or line termination. The S50 register is used to select the use of the auxiliary relay. See Auxiliary Relay Uses for details.

S50 takes one of the following values:

0	Do not use auxiliary relay
1	Use auxiliary relay for A/A1 support
2	Use auxiliary relay for spark quencher
3	Use auxiliary relay for line termination

- *U.S. terminal versions: S50 value cannot be changed.*
- *For OMNI 385 Brazilian terminals only: S50 is used to disconnect serial telephones. There are two allowable values for S50:*

0	*Do not use auxiliary relay*
1	*Use auxiliary relay for serial phone disconnect (default)*

With S50 set to 1, the default value of S51 and S52 is 50 milliseconds.

S51 Auxiliary Relay Switch In Time

The value of the S51 register determines the delay between the auxiliary relay engaging and further operations. The auxiliary relay's function is determined by the S50 register value. See Auxiliary Relay Uses for details.

S52 Auxiliary Relay Switch Out Time

The value in the S52 register determines the delay between the last operation and the auxiliary relay disengagement. The auxiliary relay's function is determined by the S50 register value. See Auxiliary Relay Uses for details.

S53 Modem Type

The value in the S53 register determines the modem type. A value of 0 sets the modem to CCITT, 1 sets the modem to BELL.

This setting affects the carrier tones and connection protocols used to establish a connection and data transfer between two modems if the terminal hardware supports both modes.

This value can also be set from the command string with the B0 or B1 command in International terminal versions.

❖ *U.S. terminal versions: S53 value cannot be changed.*

S54 Pulse-Dialing "Make" Time

The S54 register is used to set the off-hook (make) interval used when pulse dialing.

❖ *The make value is also affected by the &P command (see &P under Valid Hayes-compatible Commands). This command is valid only in international terminal versions.*

S55 Pulse-Dialing "Break" Time

The S55 register is used to set the on-hook (break) interval used when pulse dialing. The allowable values for this register are 20 through 100 milliseconds in increments

of 1 millisecond. The default is 60 milliseconds (U.S. only). The break value is also affected by the &P command (see &P under Valid Hayes-compatible Commands).

❖ *This command is valid only in international terminal versions.*

S56 Pulse-Dialing "Gap" Time

The value of the S56 register sets the pulse-dialing inter-digit "gap" time (the minimum time between each number).

S57 Ring Cycle, Minimum Frequency

The value of the S57 register sets the ring cycle minimum frequency.

S58 Ring Cycle, Maximum Frequency

The value of the S58 register sets the ring cycle maximum frequency.

S59 Dial Type

The S59 register allows the application to choose between DTMF tone dialing or one of three different pulse dialing schedules. The pulse dial schedule is the number of pulses per digit to dial. The type of pulse dialing used in the U.S. is the 10 or n type. Other countries may use one of the other two. U.S. terminal versions: only 0 or 1 are valid.

The dial types are:

0	Tone dial digit: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.
1	Pulse dial, 10 or n pulse: 10, 1, 2, 3, 4, 5, 6, 7, 8, 9
2	Pulse dial, 10 less n pulse: 10, 9, 8, 7, 6, 5, 4, 3, 2, 1
3	Pulse dial, 1 plus n pulse: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

S60 Dial Tone Check

OMNI 300 Series terminals have the ability to detect the presence of a dial tone. If S60 is set to 0, this feature is enabled. If the modem does not receive a dial tone when it attempts to dial, the unit returns response code "6", dial tone not detected.

If S60 is set to 1, this feature is disabled. The unit goes off-hook and dials, even if it does not receive a dialtone.

S61 Busy Signal Check

The modem is able to detect the presence of a busy signal after it dials. If S61 is set to 0, this feature is enabled. If the modem receives a busy signal after it dials, the modem returns response code "7".

If S61 is set to 1, this feature is disabled. The unit will not detect a busy signal.

S62 Line Check

The modem is able to detect the presence of a telephone line. If S62 is set to 0, this feature is enabled. If no telephone line is plugged in, or a parallel telephone is off-hook, the modem returns response code "90", no line, when dialing is attempted.

If S62 is set to 1, this feature is disabled. The unit goes off-hook and dials, even if the telephone line is not plugged in, or if someone is talking on the parallel telephone.

S63 Guard Tone

In the V.22 answer mode, the modem supports CCITT "guard" tones, which are enabled via the S63 register.

❖ *This feature is available only in international terminal versions.*

A value of 0 forces the modem to use no guard tone at all. A value of 1 forces the modem to use a 550 Hz guard tone. The value 2 forces the modem to use an 1800 Hz guard tone. Setting this register is equivalent to issuing a &G command.

S64 Threshold Value for DCD Loss

The S64 register is used to adjust the mechanism for detecting loss of the carrier. It is rarely necessary to change from the default setting of 8.

The modem driver samples the line approximately 16 times every 1/4 second (64 times per second). Each sample is "good" if a carrier appears to be present, based upon adequate energy levels of the appropriate frequency. If the number of "good" samples within any quarter second is less than the S64 register value, then the carrier is considered to have dropped for that quarter second. If the carrier drops for eight consecutive quarter seconds (two seconds), the modem returns response code 3, no carrier, and drops the line.

❖ *If the register is set to minimum (0), then a loss of carrier cannot be detected. The application must then be responsible for terminating the connection. In other words, this setting may be used to prevent the modem driver from dropping the line due to loss of carrier. If the register is set to maximum (15), then the modem driver is most sensitive to the loss of the carrier. This setting might be used where the application requires a "strong", continuous carrier.*

In all cases, the loss of carrier mechanism requires at least two seconds in order to drop the line. During this period, noise may be passed on as data to the application.

S65 Answer Tone Checking for V.21

The S65 register accommodates older V.21 host modems that do not raise an "answer tone" when called.

0	Require called (host) modem to generate answer tone during connection process (default)
1	Do not require answer tone

❖ *Only certain custom international terminal versions support this register.*