Cole 2 of 2

OMNI 300 Series Terminal

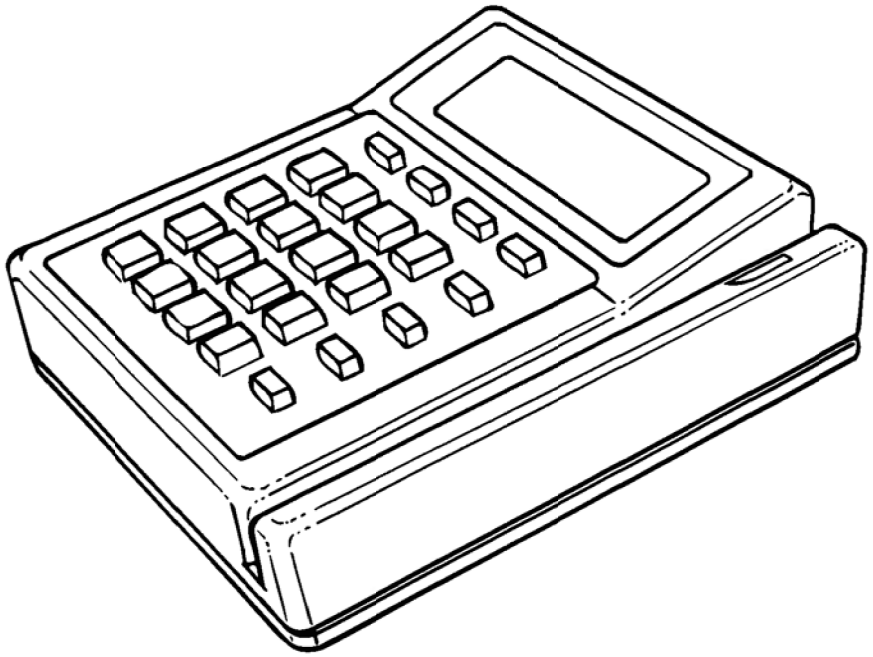
Programmer's Manual, Volume II

VeriFone Software Solutions

OMNI 300 Series Terminal

Programmer's Manual, Volume II

VeriFone Manual Part Number 12941, Revision B
Manual Revision 1.0
Included in software package, VeriFone Part Number 12860-02



VeriFone
THE TRANSACTION AUTOMATION COMPANY

IMPORTANT NOTICE

NO WARRANTY. *ALTHOUGH VERIFONE HAS ATTEMPTED TO ENSURE THE ACCURACY OF THE CONTENTS OF THIS MANUAL, THIS MANUAL MAY CONTAIN ERRORS OR OMISSIONS. THIS MANUAL, INCLUDING WITHOUT LIMITATION THE SOFTWARE PROGRAM EXAMPLES CONTAINED HEREIN, IS SUPPLIED "AS-IS," WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.*

LIMITED LIABILITY. *IN NO EVENT SHALL VERIFONE BE LIABLE FOR ANY INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES (INCLUDING DAMAGES FOR LOSS OF BUSINESS, PROFITS OR THE LIKE) EVEN IF VERIFONE OR ITS REPRESENTATIVES HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.*

PRODUCT CHANGES. *VERIFONE RESERVES THE RIGHT TO CHANGE, UPDATE, OR MAKE OBSOLETE ANY OMNI 300 SERIES DIAL TERMINAL AT ANY TIME AND WITHOUT NOTICE.*

OMNI 300 Series Terminal Programmer's Manual, Volume II
VeriFone Part Number 12941, Revision B
Manual Revision Number 1.0
Included in software package, VeriFone Part Number 12860-02

PRINTED: September, 1994

VeriFone, Inc.
Three Lagoon Drive
Redwood City, CA 94065
TELEX: 5106007959 VERIFONE

CHAPTER 10
OMNI Peer-to-Peer LAN

CHAPTER 11
Library Functions

APPENDIX A
International Certification Restrictions

APPENDIX B
Error Return Codes

APPENDIX C
Model Specifications

APPENDIX D
OMNI 460 Printer Programming

APPENDIX E
VeriFone Fonts

APPENDIX F
Portability Guidelines

Index, Volume I and Volume II

C H A P T E R

10

OMNI Peer-to-Peer LAN

The OMNI 3xx Local Area Network (LAN) is a peer-to-peer network that allows all terminals on the LAN equal access to one another. The peer-to-peer LAN typically comprises an OMNI 4X0 terminal acting as a communications "gateway" to the outside world and up to 31 networked OMNI 3X5 terminals.

LAN Hardware

The LAN transmission medium consists of a three-wire cable. The wires are LAN +, LAN -, and LAN ref. The terminals are each connected to these wires in parallel. Each terminal on the LAN has both send and receive capability to all other terminals. This is accomplished by implementing a Character Sense, Multiple Access/ Collision Avoidance (CSMA/CA) scheme. In this scheme, each terminal is allowed a time window of access to the LAN.

The signaling method for the LAN is RS-485. The LAN BUS length maximum is 4000 feet. Additionally, drop wires of up to 12 feet may be used to connect a terminal to the BUS with no detrimental effect on LAN

performance. The BUS must be electrically terminated at each end for optimum performance. This termination is provided by special LAN cables which access a termination resistor in the terminal. These special cables must be used for the end terminals.

LAN Operating System Software

The OMNI LAN is functionally defined in layers 2, 3, and 4 of the OSI architectural standard for networks. These three components are implemented in the OMNI 3X5 LAN operating system.

O.S.I. MODEL	OMNI 3X5 LAN MODEL
Application (layer 7)	TXO Application
Presentation (layer 6)	Undefined
Session (layer 5)	Undefined
Transport (layer 4)	Protocol Driver
Network (layer 3)	Protocol Driver
Data Link (layer 2)	Protocol Driver
Physical (layer 1)	Wire

The protocol driver formats data packets for transmission by adding header information and a CRC checksum before sending them to the port driver (CSMA/CA).

On the receiving end, packets are validated and processed based on their 'type'. Some packets (e.g., NORM) are passed all the way up to the application, while other types, e.g., ACKs and NAKs, are used by the operating system and are transparent to the application. Before sending a packet to the application, the operating system validates and then strips off the checksum.

LAN Terminal Installation Procedures

Installing an OMNI 3X5 LAN terminal on the LAN involves two steps.

1. Physically connecting the terminal to the LAN and powering up the terminal. In the field, this normally requires connecting the terminal's RJ-11 line cord to the LAN trunk connection. In a development configuration, each terminal on the LAN can be connected on a BUS using standard RJ-11 cabling with proper terminators. See *LAN Hardware*, page 10-1, for configuration details.

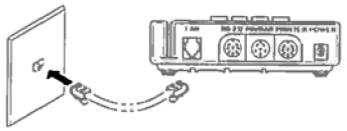


Figure 10-1: OMNI 385 LAN Connection (Field Installation)

2.) Downloading the application either from a LAN download server or via the terminal's serial printer port from a PC or another terminal. This chapter describes in detail the various downloading methods available to the application.

OMNI 3X5 LAN terminals use a LAN port with standard RJ-11 connectors common to telephone and modem technology. It supports a proprietary VeriFone Peer-to-Peer LAN communications protocol (CSMA/CA)—well suited to the transaction automation environment.

LAN Data Exchange

LAN messages are exchanged on the LAN using a data packet, which consists of an 18-byte header, 0 to 1500 bytes of binary data and a 2-byte checksum. The packet header is defined as `struct packet_header` in `<lan.h>`.

Control	Name	Label (<lan.h>)	Range	Size (bytes)
ignored	Source address	src_reserved	—	1
ignored		src_net	—	1
ignored		src_socket	—	1
firmware		src_addr	1 – 32	1
ignored	Destination address	dst_reserved	—	1
ignored		dst_net	—	1
ignored		dst_socket	—	1
application		dst_addr	1 – 32, 254	1
firmware/ application	Sequence number	tx_seq	0 – 0xFFFFFFFF	4
ignored	Protocol	protocol	—	2
application	Application message type	app_type	app.-defined	1
firmware/ application	Message type	type	0x0000–0x0082	1
firmware	Data length	dg_length	0 – 1500	2
application	Data	—	binary data	< 1500
firmware	CRC-16	—	0 – 0xFFFF	2

Data Packet Fields

This section describes the fields within a data packet that are used by the firmware or the application.

Source Address—Byte #4

Identifies terminal sending the packet.

Defined as `unsigned char src_addr;`

Range of values: 1 – 32

- *The first three bytes in the source address field are not currently used and are reserved for future use. The value of* `src_addr` *is obtained from the* `*LAD` *entry in the CONFIG.SYS file.*

Destination Address—Byte #8

Identifies terminal to receive packet or specifies a send to all terminals—BROADCAST.

Defined as `unsigned char dst_addr;`

Range of values: 1 – 32, 254 = BROADCAST

- *The first three bytes of the destination address field are not currently used and are reserved for future use. The value of this field is controlled by the application.*
- *If* `dst_addr` *is set to 254, the packet is sent to all terminals on the LAN (BROADCAST type message). Although all terminals receive the broadcast destination packets, they do not need to send any acknowledgment (ACK or NAK) to the packet.*

Sequence Number—Bytes #9–12

Identifies order of data packets from sending terminal.

Defined as `unsigned long tx_seq;`

Range of values: `0 - 0xFFFFFFFF`

See *Sequence Numbers* later in this section.

❖ *The application may write to the* `tx_seq` *field if the* `type` *field is DOWNLOAD or* `dest_addr` *is BROADCAST. Otherwise, the operating system controls the sequence field and overwrites any value set by the application. If an application-generated BROADCAST message is the first message after LAN initialization, the application must manually set the sequence number to 1.*

Application Message Type—Byte #15 and Message Type—Byte #16

This two-byte `type` field has two subparts.

Application Message Type: Defined as `unsigned char app_type;` May be used by the application to create a set of internal message types. (This sub-field is ignored by the firmware.)

Message Type: Defined as `unsigned char type;` Used by both the application and the firmware. May be set by the application to `NORM`, `DOWNLOAD` or `DIAG_REQ`.

Value	Type	Description
0x0081	NORM	Data contains application data
0x0082	DOWNLOAD	Data contains download file request data
0x0004	DIAG_REQ	Diagnostic count request

If the application uses a value other than the above, the firmware overrides this field with `type = NORM`.

Value	Type	Description
0x0000	I_M_HERE	Data contains resource definition
0x0001	INSTALL	Fresh terminal installation request
0x0003	PROTO_RESET	Sending terminal has been reset
0x0006	ACK	Terminal has received message correctly
0x0015	NAK	Terminal has received corrupted message
0x0080	DIAG_RESP	Data contains diagnostic count

Length—Bytes #17–18

Specifies the length of the data field.

Defined as `unsigned int dg_length;`

Range of values: 0–1500

- *This value does not include the 18-byte header or 2-byte CRC trailer. Since the total packet length cannot exceed 1520 bytes, this field will never be greater than 1500. This field is controlled by the operating system.*

Packet Data

Following the header is the actual data field of the packet (if any—some packets, for example, an ACK packet, contain no data other than the packet type). The data field is free form and may contain from 0 to 1500 bytes of binary data.

CRC-16

This 2-byte field is the Cyclic Redundancy Code created and used by the firmware and is transparent to the application.

Protocol Message Exchange

This section describes various OMNI LAN message exchanges in an error free environment.

- *NORM, DOWNLOAD, and DIAG_REQ type messages can be sent by the application in a BROADCAST mode (destination address = 254). A BROADCAST type packet is processed by all active LAN terminals, although the destination terminals do not acknowledge (ACK/NAK) the packet.*

NORM Message Exchange

Any terminal transmitting a NORM message type expects to receive an ACK from the destination terminal. Any terminal that receives a NORM message type with a DST_ADDR equal to its own terminal address will respond with an ACK message type if the checksum is correct.

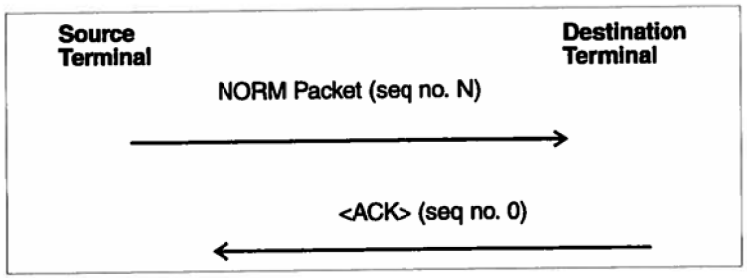


Figure 10-1: NORM Message Protocol

DIAG_REQ Message Exchange

A terminal's response to a DIAG_REQ message is to send the diagnostic count data to the requesting terminal. The data packet consists of 68 2-byte integers. See *LAN Diagnostic Counts* later in this section for additional information.

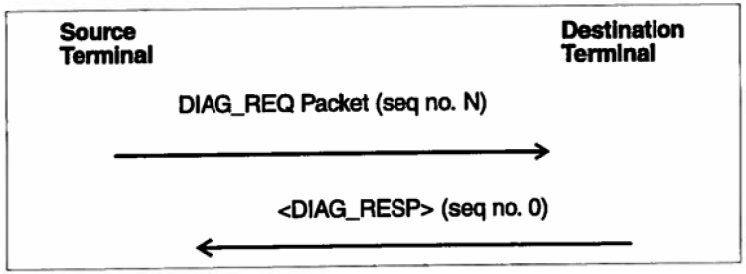


Figure 10-2: DIAG_REQ Message Protocol

Download Message Exchange

This topic is descussed in detail later in this chapter under *LAN Downloading*.

I_M_HERE Message - Node List Update

Every 64 seconds, each terminal transmits an I_M_HERE type message to all terminals (BROADCAST destination) on the LAN. The data section of this message contains the resource definition (1 byte) and the LAN high address (1 byte). The LAN high address is for information purposes only and is not processed by any terminal.

As each terminal receives this message, the firmware updates its internal node list, which contains the application resource definition for each terminal. If the terminal does not receive an I_M_HERE packet from a terminal within approximately 195 seconds, that terminal's resource definition is dropped from the node list.

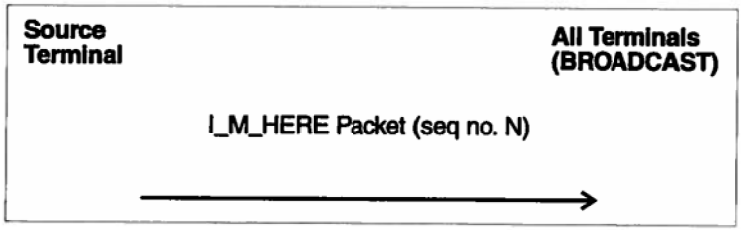


Figure 10-3: I_M_HERE Message Protocol

Protocol Reset Message

A PROTO_RESET type message is always sent as a BROADCAST type packet. It indicates that the sending terminal's application has just set or reset the LAN's `Opn_Blk` structure with an `ioctl()` call using `SetCtrl | 0`. In response to this message, receiving terminals reset their transmit and receive sequence numbers for the sending terminal.

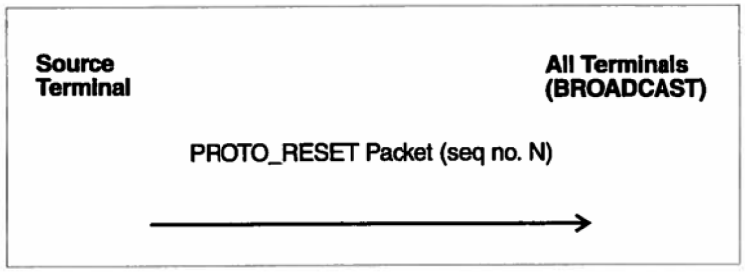


Figure 10-4: PROTO_RESET Message Protocol

Protocol Driver Error Recovery

OMNI 300 Series use several error recovery methods to handle corrupt packet data, download errors, lost data and squence number errors.

Corrupt Data

If the destination terminal receives a data packet whose CRC-16 entry does not match the terminal's CRC calculation, it sends a NAK to the sending terminal.

In all message type transmissions except DOWNLOAD, the NAK response will return the sequence number (TX_SEQ field) of the offending message. If a NAK is transmitted due to a checksum error, the original sending terminal will re-transmit the packet if the TX_SEQ in the NAK packet matches the TX_SEQ in the original packet.

Corruption may occur in the header in several ways—a packet may include the wrong destination (e.g., DST_ADDR field is corrupt), a NAK could be returned to the wrong destination (SRC_ADDR is corrupt) or the TX_SEQ number could be corrupt.

If the packet is received by the wrong terminal (DST_ADDR is corrupt), and the NAK is returned, the original packet will be re-transmitted.

If the SRC_ADDR is corrupt, the ACK/NAK will be returned to the wrong terminal. The terminal that received the ACK/NAK will not respond because it will not have an outstanding message with the same sequence number destined for the terminal that sent

the ACK/NAK. The original sender will re-transmit the packet, as an ACK/NAK was not received.

If the sequence number is wrong, the NAK that is returned will be ignored due to the TX_SEQ mismatch. The original sender will time-out as no ACK was received and will re-transmit the packet.

Download Error Recovery

Error recovery for downloads follows the download protocol covered later in this section under LAN Downloading.

Lost Data

Recovery from a lost packet is simple: if an ACK response is not sent within an application-defined transmit time-out period, the packet is re-transmitted. The packet will be re-transmitted up to a maximum number of times defined by the application in the `Opn_Blk` structure.

If an ACK packet is lost, the original terminal will re-transmit the message. The receiving terminal will recognize that the message is identical to the last received message from that terminal, throw the message away and reply with an ACK.

Sequence Numbers

There are a few scenarios in which a sequence number is out of order. Generally, if a terminal receives more than one packet with the same sequence number, the second packet and all subsequent packets with the same sequence number are ACKed but are thrown away.

Another sequencing problem arises when one terminal is rebooted in the middle of a transaction. When a terminal powers up, it sends out a PROTO_RESET packet. All terminals on the LAN reset the sequence number for this terminal to 0. However, if a packet is already queued for a retry or requeued on the reject queue, the packet

sequence number will not be reset. To keep the two terminals in sync, when the terminal sending the PROTO_RESET packet receives a new packet, it ACKs the packet, passes the packet to the application, and sets its sequence number to match the incoming packet.

For example, terminals #1 and #2 are in the middle of a transaction. Terminal #1 sends a packet with sequence number 3. At this point, terminal #2 is rebooted and sends out a PROTO_RESET packet. Ordinarily, all terminals will reset their sequence numbers to 0. Since terminal #1 never saw an ACK from terminal #2, it will place the packet with sequence number 3 in the retransmit queue. The packet is re-transmitted with sequence number 3. Terminal #2 ACKs the packet, passes it to the application, and sets its sequence number to 3 (to match the packet received from terminal #1).

LAN-related Buffer Management

OMNI 300 Series terminals use a shared pool of buffers, each 254 bytes in length. The default number of buffers is 4 and may be set up to 32 at the time of application download via the `*B` parameter in the CONFIG.SYS file. This buffer pool is shared by all I/O functions, including the LAN. To avoid running out of buffers, pay prompt attention to pending received LAN messages and rejected transmissions. Use the Data Packet Received Trap (53) to promptly `read()` pending input messages and use the Exception/Event Trap (52) to detect failed transmissions and to promptly perform one of the following:

- purge all reject messages with `SetCtrl | 15 ioctl()` variant,
- read reject messages one by one with `ioctlc()` command, or
- requeue all current reject messages with the `SetCtrl | 14 ioctl()` variant.

Prompt handling of received and rejected messages will help ensure an adequate number of buffers are available for the intended application and environment. There is

no formula for dictating an adequate buffer count, as the number of terminals on the LAN, message traffic, and handling I/O and buffer status must be taken into consideration. We generally recommend specifying the maximum number of buffers possible (up to 32). The trade off is that increasing the buffer count reduces the remaining program space accordingly.

Remote Diagnostics

Applications may request diagnostic information from any other terminal on the LAN by sending a DIAG_REQ packet. This packet is intercepted by the LAN driver on the receiving terminal and is not seen by the application. The response is a DIAG_RESP packet, whose data section contains 69 2-byte integer values as defined in the table on the following page. Note that neither the DIAG_REQ nor DIAG_RESP packets are ACKed by the receiver.

LAN Terminal Diagnostic Counts

Diagnostic	Count
User-defined counts	0–19
Physical key presses	20
Power-on cycles or System restarts	21
[CLEAR] exits from System Mode	22
ZONTALK downloads attempted	23
Direct uploads attempted via [*] key in system mode	24
Direct downloads attempted via [#] key in system mode	25
Normal beeps	26
Error beeps	27
Barcode reads attempted	28
Barcode scans with errors	29
Card reads attempted	30
Card reads with errors	31
Counts 32–45 and 58 are not relevant to the 3X5 LAN terminal which lacks an internal modem.	—
ACKS sent on COM1 port	46
NAKS sent on COM1 port	47
ACKS received on COM1 port	48
NAKS received on COM1 port	49
ACKS sent on PIN Pad/Bar Code port	50
NAKS sent on PIN Pad/Bar Code port	51
ACKS received on PIN Pad/Bar Code port	52
NAKS received on PIN Pad/Bar Code port	53
ACKS sent on COM3 port	54
NAKS sent on COM3 port	55
ACKS received on COM3 port	56
NAKS received on COM3 port	57
ACKS sent on LAN port	59
NAKS sent on LAN port	60
ACKS received on LAN port	61
NAKS received on LAN port	62
Transmit timeouts on LAN port	63
Runt packets† received on LAN	64
Reserved for future use	65–68

†*Runt packet: Header is less than minimum required size and is invalid (not processed).*

LAN CONFIG.SYS Settings

This section lists all CONFIG.SYS variables required for LAN installation, downloading, and optional operations. These variables must be in the terminal prior to downloading and application startup. If not present, the operating system will automatically prompt the user to enter the required values.

The CONFIG.SYS variables are listed below in the order in which they are prompted if they are either not set or set to invalid values. Once entered, these values are maintained even if the terminal loses power (power cycle).

CONFIG.SYS LAN Variables

Variable	Range	Description
***LBR**	4, 5, 6	LAN Baud Rate (4=4800, 5=9600, 6=19,200)
***LAD**	1-32	Terminal address
***LDS**	1-32, 254	LAN Download Server Address (see *Notes*)
***LHA**	2-32	LAN high addr. (highest address on LAN—see *Notes*)
***LTW**	1, 2	LAN Timing Window
***ZT**	numeric	Application serial number
***ZA**	string	Application file name without .DLD file extension
***LFP**	F or P	Download type "Full" or "Partial" (see *Notes*)
The following variables are not required. (The operating system will not prompt for values prior to startup.)		
***LAN**	0 (alpha)	LAN Protocol
***LZF**	A	Download-all type download (see *Notes*)

- *When the *LDS parameter is set to 254, a terminal-initiated download is sent as a BROADCAST message and the first server to respond performs the download.*

☞ *The *LFP CONFIG.SYS variable is required for a LAN download. In an empty terminal, *LFP is not set, or is set to an invalid value, the terminal enters System Mode and the user is prompted to enter the valid value. The prompting sequence will display* `PARTIAL OR FULL?` *and* `PART=. FULL=FUNC` *in continuous alternation until the [*] or [FUNC] is pressed.If it is set to a valid value, the specified type of download is requested without prompting for the download type. This flag has no effect on application-initiated downloads.*

☞ *The *LZF CONFIG.SYS variable is optional. When *LZF = A, the terminal will request a download to all terminals on the LAN instead of requesting a download for just itself. If it is not set, or set to an invalid value, the terminal requests a download for itself.*

LAN Function Calls

The OMNI Peer-to-Peer LAN supports all of the standard I/O related functions—`open()`, `read()`, `write()`, `ioctl()`, and `close()`—as well as the higher-level functions `put_lan_config()`, `get_lan_config()`, and `ioctlc()` (closely related to the standard `ioctl()` function).

open()

Prepares the LAN port for use by the application.

☞ *The LAN port cannot be configured before this call is made. Use* `put_lan_config()` *to configure all LAN parameters, except *LAD, the terminal address. The *LAD parameter may be entered through the keyboard or as a downloaded parameter.*

```
#include <io.h>

hLAN = open ("/dev/lan", 0);
int hLAN;
```

The first parameter of the call is not case sensitive, and the second is ignored. The `open()` command initializes all of the LAN queues (messages received, rejects, pending transmits) and the buffer space, and sets the default exception event trap mask to detect only the `XMIT_FAILURE` condition.

This call will always succeed, returning a valid handle, unless the first parameter of the call is incorrect.

put_lan_config()

Performs LAN port configuration and initialization per the specified terminal and protocol parameters. The prototype is:

```
#include <lan.h>

result = put_lan_config (hLAN, term_parms);
int hLAN, result;
LAN_TERM_PARMS *term_parms;
```

- *The LAN port must be opened prior to this call. This function must be called before the application can use other I/O functions (*`read()`, `write()`, `ioctl()`, *etc.).*

- *After* `put_lan_config()` *is called, the firmware sends a PROTO_RESET message to all other LAN terminals informing them that the terminal's parameters have been reset.*

The data structure `LAN_TERM_PARMS` (defined 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;
```

This function returns a 0 on success, or a -1 with `errno` set to indicate the type of error. The function can fail if either the underlying `ioctl()` call it makes to the port fails, in which case the `errno` returned is that of the `ioctl()`, or if the `*LAD` terminal address parameter is not found in the CONFIG.SYS file, in which case `errno` is set to `EINVAL`.

Terminal Parameters

The four terminal parameters used in the `LAN_TERM_PARMS` data structure are controlled by the application; their values may vary from terminal to terminal and may be modified over time as the application changes. There are no implied defaults for these parameters. When this function is called it will use the value found in the currently defined `term_parms`. The description of these terminal parameters are described on the following page.

`receive_timeout`

The `receive_timeout` parameter specifies the amount of time (1–10 seconds) the protocol will wait for an ACK from the destination terminal after attempting a message transmission over the LAN. After that number of seconds, the transmission is considered a failed attempt. This parameter must be in the range of 1 to 10.

`resource_definition`

The `resource_definition` lets the application associate the address of a terminal with a specific LAN resource, such as a print server or host gateway.

`transmit_retries`

The `transmit_retries` parameter specifies how many additional times a terminal will attempt to transmit a message when no ACK is received. This value must be in the range of 1 to 10. For example, a terminal with `transmit_retries` set to 3 will attempt an unacknowledged transmission a total of 4 times (3 retries) before placing the message in the reject queue.

`throttle`

The `throttle` parameter must be set to either 0 (off) or 1 (on). When turned on, a terminal waits longer than usual to send a message on the LAN. Thus it has the effect of slowing down transmission.

In typical use, it is turned on in most terminals to prevent low-address terminals from taking too much of the LAN's resources under heavy loads, but may be turned off for such individual terminals as gateways or servers so their input is increased. Client (transaction) terminals should almost always leave their throttle on.

Protocol Parameters

The LAN also uses four protocol parameters, which are not under control of the application. These are configured during LAN installation where their values are set in each terminal's CONFIG.SYS file. The `put_lan_config()` function checks CONFIG.SYS for values associated with each of these protocol parameters. If no value is present, it will select defaults for all but the `*LAD` terminal address parameter . If no `*LAD` value is present, `put_lan_config()` returns a -1 with an `errno` set to `EINVAL`.

get_lan_config()

Retrieves the current parameters used to configure the LAN. The prototype is:

```
#include <lan.h>

result = get_lan_config(hLAN, term_parms,
                        prot_parms);
int hLAN, result;
LAN_TERM_PARMS *term_parms;
```

See page 10-17 for the data structure `LAN_TERM_PARMS`.

The data structure LAN_PROT_PARMS (defined 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;
```

This function returns a result of 0 if the underlying ioctl() call to retrieve the parameters succeeds, or -1 with errno set according to the underlying ioctl() error.

Protocol Parameters

get_lan_config() returns the values of both the four terminal parameters (discussed earlier) and the four protocol parameters to the application. The functions of the protocol parameters are as follows:

rate

The rate parameter gives the LAN baud rate as set via the *LBR parameter in the CONFIG.SYS file. This value must be identical for all terminals on the LAN.

address

The address parameter provides the terminal's LAN address as set via the *LAD parameter in the CONFIG.SYS file.

highest_address

The highest_address parameter is the highest expected (or actual) address of any terminal on the LAN as set via the *LHA parameter in the CONFIG.SYS file. This value must be identical for all terminals on the LAN.

timing_window

The timing_window parameter determines the rate of data throughput—1 (fast) or 2 (slow)—and is set via the *LTW parameter in the CONFIG.SYS file. The fast setting is "normal" and maximizes LAN throughput. The slow setting may be used for LANs operating in adverse environments—ones which may experience greater than normal electrical interference (noise). This value must be identical for all terminals on the LAN.

read()

Transfers the contents of the internal device buffer to the application's buffer. The prototype is:

```
#include <lan.h>

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

Each `read()` transfers `size` bytes of the pending message from the input queue to the application `buffer`, and returns as `bytes_read` either the actual number of bytes transferred, 0 if no input data was pending on the LAN port, or a -1 if an error occurs (setting `errno` accordingly). An error will occur if a read is attempted before the LAN is opened and configured, or after it is closed. If `size` is less than the actual length of the input message, the message is truncated and leftover bytes are lost. Thus an application's `buffer` and `size` must anticipate the largest possible message.

Since the maximum message length is 1518 bytes and buffers in the buffer pool are 254-bytes long, a message may occupy multiple buffers. This is handled transparent to the application. Note that in the event there are insufficient buffers to handle the competing I/O demands of the LAN, only part of a message may be received. For this reason it is not advisable to "spin" waiting on the receipt of a message, as in the statement:

```
while (read (...) == 0);  /* Avoid this kind */
                          /* of read().      */
```

Instead, the application should use the data packet received trap (Trap 53) to indicate the receipt of a complete data packet and trigger the `read()` action.

write()

Transfers the contents of the application's buffer to the LAN's transmit queue. The prototype is:

```
#include <lan.h>

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

Each `write()` transfers `size` bytes from the application's `buffer` to the transmit queue, and returns control immediately to the application. The returned `bytes_written` will be the actual number of bytes moved to the transmit queue. The contents of the `buffer` must be a correctly formed LAN message comprising an 18-byte header and the variable length data field. The destination address must be supplied by the application.

The minimum allowable message size is 18 bytes (not counting the CRC, which is either generated or alternately validated and stripped off by firmware, and is thus transparent to the application). Messages less than 18 bytes return `bytes_written` as -1 and `errno` set to `EINVAL`. Also, if the `write()` is attempted before the LAN is opened and configured or after it is closed, a -1 will be returned.

The maximum allowable message size is 1518 bytes, including the header. Any attempt to write a larger message will result in a "streaming line" error, which has serious consequences and should be avoided. A streaming line error usually indicates faulty hardware. All terminals detecting this condition will block the LAN port, preventing further input and output. However, previously received "pending" messages will still be available to be read. After a streaming line error, the LAN needs to be reopened and reconfigured by the application.

Once a message is moved into the transmit queue, it will go out over the LAN under firmware control according to its order in the queue, and will be ACKed by the intended

destination terminal. If this ACK is not received after the specified number of retries, the message is transferred to the reject queue.

Only one message per destination terminal may be queued at a time. The message will stay in the queue until it has been successfully delivered (i.e., ACKed by the receiving terminal). While the message is pending, any other attempts to write to the same destination terminal will be blocked and the `write()` command will return a `bytes_written` result of -1 and set `errno` to `EBUSY`.

If the application sends a message to an address within the LAN high address parameter but no terminal with that address exists on the LAN, `write()` will look normal; however, the transmission will fail and the packet is rejected. If the application sends a message to a terminal address greater than the LAN high address, the `write()` fails and returns a `result` of -1 with `errno` set to `EINVAL`. A terminal may attempt to `write()` a message to its own address. However, it will not see a message from its own address, and the message will go to the reject queue.

☞ *Attempts to* `write()` *to an unopened LAN will fail. A* `write()` *will also fail (with a* `result` *of -1 and* `errno` *set to* `ENOSPC`*) if there are an insufficient number or no available buffers for the given message. Try to avoid developing these situations. More information is given later in this section under LAN-related Buffer Management.*

TXO Trap 52 (XMIT_FAILURE) is provided to signal that the transmit has failed. Normally, the TXO application should arm this trap and process the failed `write()` attempt in the guard function.

ioctl()

Allows the application to perform various control and status services. The prototype for this function is:

```
#include <lan.h>

result = ioctl (hLAN, command, buffer);
int result, hLAN, command;
char *buffer;
```

The exact functionality of this call depends on the value of `command`. Two macros are defined, `SetCtrl` and `GetCtrl`, to help differentiate between status functions and control functions, respectively. The table on the following page lists the available command options and briefly describes each.

Command		Command Description
SetCtrl	14	Requeue reject queue
SetCtrl	15	Empty (purge) reject queue
SetCtrl	29	Set exception/event trap mask
GetCtrl	0	Get current LAN port status bytes
GetCtrl	2	Used with ioctlc() to read reject queue
GetCtrl	10	Read exception/event trap mask
GetCtrl	11	Read exception/event trap status
GetCtrl	12	Read LAN node list
GetCtrl	13	Read LAN ACK list

In general, on success `result` will contain a 0. Expected non-zero successful returns are specified in the following descriptions of individual `SetCtrl` or `GetCtrl` variants. On error, `result` will contain `-1` and `errno` will be set to either `EINVAL`, for an invalid command, or `EBADF` if the LAN is not open or not initialized.