

AC31
INTELLIGENT INTERFACE ADAPTER
FOR
THE OPTOMUX NETWORK

Form 168.6

August 1989

This technical document describes the features, specifications, and operations of the product.

For specific wiring connections and operational specifications of I/O modules and mounting racks, please refer to the *I/O SYSTEMS COMPONENTS* manual, Form #132.

The information in this manual has been carefully checked and is believed to be accurate; however, no responsibility is assumed for possible inaccuracies or omissions. Specifications are subject to change without notice.

Opto 22 warrants all its products to be free from defects in material or workmanship for 24 months from the manufacturing date code.

This warranty is limited to the original cost of the unit only and does not cover installation labor or any other contingent costs.

OPTOMUX is a trademarks of OPTO 22.

Table of Contents AC31

INTRODUCTION	1-1
INSTALLATION AND WIRING	1-2
AC31/SLAVE/HOST WIRING DIAGRAM (Using "D" Shell Connector)	1-8
AC31/SLAVE/HOST WIRING DIAGRAM (Using Screw Terminals)	1-9
AC31/SLAVE RS232 CONNECTOR WIRING	1-10
COMMAND STRUCTURE	1-11
DATA STRUCTURE	1-13
RETURN MESSAGE STRUCTURE	1-14
ERROR CODES	1-16
QUICK REFERENCE	2-1
COMMAND DESCRIPTIONS	3-1
POWER UP CLEAR	3-2
RESET	3-3
SET TURNAROUND DELAY	3-4
RETRANSMIT LAST RESPONSE	3-5
SET END OF TEXT CHARACTER	3-6
RETURN DEVICE TYPE	3-7
READ BINARY STRING	3-8
WRITE BINARY STRING	3-9
GET TRANSMIT STATUS	3-10
READ ASCII STRING	3-11
READ NUMBER OF ASCII CHARACTERS	3-12
OUTPUT MESSAGE BUFFER TO SLAVE	3-13
SET DEVICE PORT CONFIGURATION	3-14
ENABLE CONTROL S/CONTROL Q PROTOCOL	3-15
DISABLE CONTROL S/CONTROL Q PROTOCOL	3-16
RETURN NUMBER OF CHARACTERS WAITING	3-17
WRITE ASCII STRING	3-18
WRITE ASCII STRING AND WAIT FOR RESPONSE	3-19
SET TURNAROUND TIME-OUT	3-20
WRITE A MESSAGE TO A MESSAGE BUFFER	3-21
READ MESSAGE BUFFER	3-22
APPENDIX A - BASIC Subroutines	A-1
APPENDIX B - ASCII Chart	B-1

1.1 INTRODUCTION

The AC31 is an addressable interface for the OPTOMUX™ bus.

Many devices incompatible with the OPTOMUX™ bus protocol can be made compatible by using an AC31. Serial printers, terminals, barcode readers and asynchronous devices can now be talked to.

The AC31 can perform the three following functions:

- *buffers asynchronous messages until the host requests them*
- *adds addressability to dumb devices (printers, terminals, etc.)*
- *allows communication with devices that have a protocol that conflicts with the OPTOMUX™ protocol.*

NOTE:

Through out this manual, a carriage return character is denoted by the italicized characters *cr*.

1.2 INSTALLATION AND WIRING

The following steps will describe how to properly configure and install your AC31.

STEP 1: Communication Cables

All of the communications terminals on the barrier strip are duplicated on the 9-pin D shell connector. This gives you a choice as to which type of connection to use. Refer to the wiring diagrams following this section for information on cable connections and pin assignments. The wiring diagrams show host to AC31 cabling using either the 9-pin D shell connector or the terminal barrier strip. If you are using an OPTO 22 RS-422 card (model AC24, AC422, or AC32), please note that the wiring is not a straight through, pin for pin connection between the two 9-pin D shell connectors.

For each port, use a 22 gauge (minimum) cable with two twisted pairs. Connect the wires as show in the wiring diagrams.

Double check your wiring after it is complete, then continue with the next step.

STEP 2: Setting the Jumpers

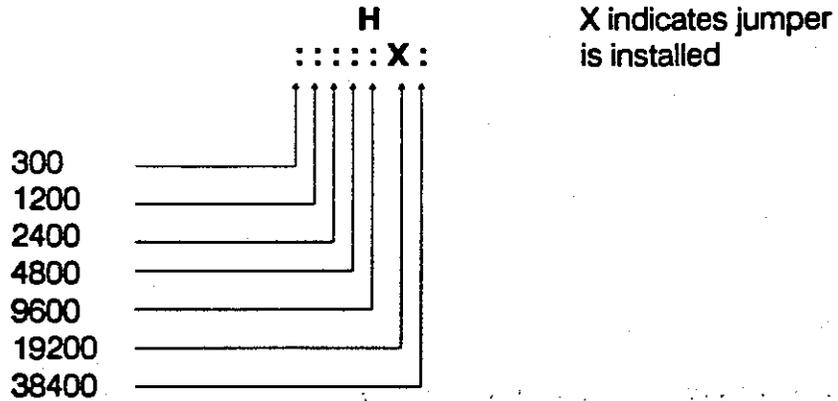
There are two groups of jumpers for selecting the baud rate; one group for each communications port. The group of jumpers corresponding to the host port is labeled **H**, the group corresponding to the slave port is labeled with a **S**.

The two communication ports operate independently. A different baud rate may be specified for each port. Only install one jumper per group.

Install the jumper for the desired baud rate on the group **H** block for communications with a host device.

Example:

By installing the jumper in the position marked 19.2K, the baud rate for the host port is set to 19200 baud



After installing the jumper for the host port baud rate, install the jumper for the slave port (group S).

Installing the Group A Jumpers

The group A jumpers are used to add termination resistors and pull-up or pull-down resistors to each communication pair.

The following examples illustrate which jumpers should be installed for specific applications. In order to properly balance the communications lines, the termination resistors should be installed only on the devices at each end of the link. Only one device on the entire link should have the pull-up and pull-down resistors. Refer to the Communications Jumpers Schematic for a diagram of each jumper and its corresponding resistor.

Example 1:

One AC31 connected to a host device (a computer with an RS-422 communications interface).

Group A							
1	2	3	4	5	6	7	8
X	X	X	X	X	X	X	X

X indicates jumper is installed

Example 2:

Several AC31s connected in a multidrop configuration (communication lines paralleled).

Devices on each end of the physical link should have termination resistors installed. One device should have the pull-up and pull-down resistors installed (preferably the host).

AC31 on the end of the link

Group A
1 2 3 4 5 6 7 8
X X X X X X X X

X indicates jumper is installed

AC31s NOT on the end of the link

Group A
1 2 3 4 5 6 7 8
X X : : X : X :

X indicates jumper is installed

STEP 3: Configuring the Host and Applying Power to AC31

Refer to the Operations Manual of your host terminal or computer for configuring and initializing the host's serial port. The host should be set up with the following parameters:

BAUD RATE:	300, 1200, 2400, 4800, 9600, 19200 or 38400
START BITS:	1
STOP BITS:	1
DATA BITS:	8
PARITY:	NONE

After the host is configured, apply power to AC31.

NOTE: If parity or the number of data bits on your host is incorrect, the AC31 will not communicate with the host computer.

AC31 JUMPER DESCRIPTIONS

The AC31 contains several onboard jumpers which must be configured for proper operation. The jumpers determine the communication baudrates, and the termination of communication lines. The following table describes each jumper setting:

GROUP A JUMPERS

A1	Pull up resistor on "FROM SLAVE +" line.
A2	Pull down resistor on "FROM SLAVE -" line.
A3	Pull up resistor on "FROM HOST +" line.
A4	Pull down resistor on "FROM HOST +" line.
A5	Terminating resistor on "FROM SLAVE" line.
A6	Terminating resistor on "FROM HOST" line.
A7	Terminating resistor on "TO SLAVE" line.
A8	Terminating resistor on "TO HOST" line.

- *Jumper in - connects resistor to line*

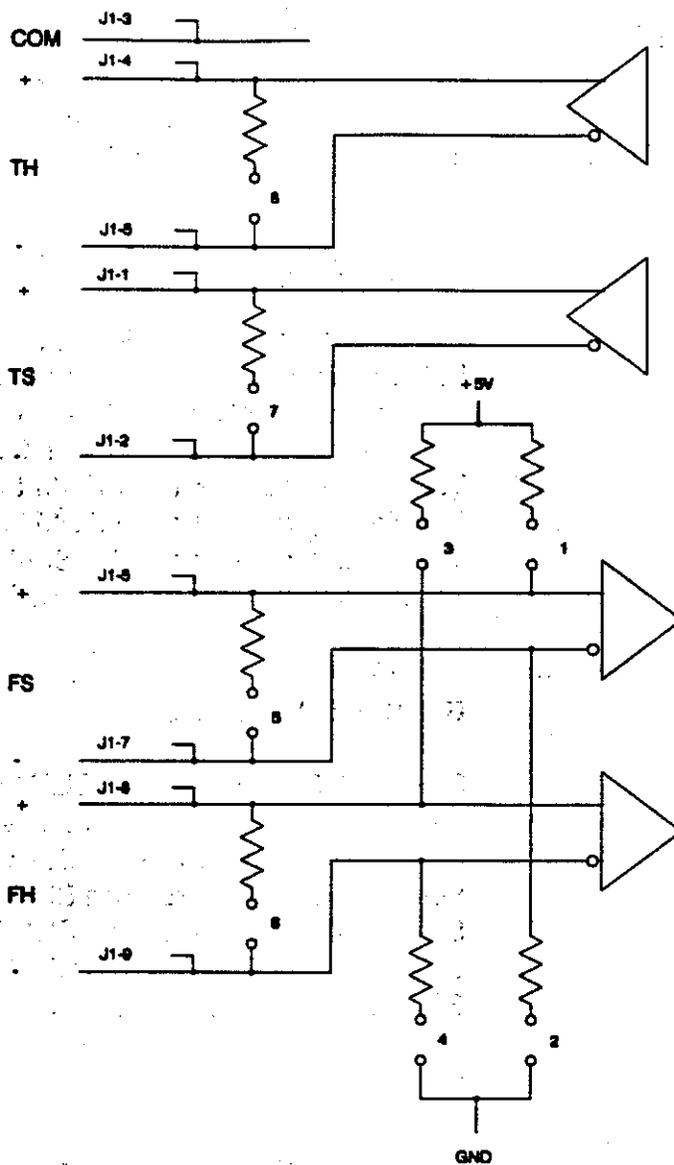
GROUP S JUMPERS

300	Selects 300 BAUD rate on SLAVE port
1200	Selects 1200 BAUD rate on SLAVE port
2400	Selects 2400 BAUD rate on SLAVE port
4800	Selects 4800 BAUD rate on SLAVE port
9600	Selects 9600 BAUD rate on SLAVE port
19.2K	Selects 19200 BAUD rate on SLAVE port
38.4K	Selects 38400 BAUD rate on SLAVE port

GROUP H JUMPERS

300	Selects 300 BAUD rate on HOST port
1200	Selects 1200 BAUD rate on HOST port
2400	Selects 2400 BAUD rate on HOST port
4800	Selects 4800 BAUD rate on HOST port
9600	Selects 9600 BAUD rate on HOST port
19.2K	Selects 19200 BAUD rate on HOST port
38.4K	Selects 38400 BAUD rate on HOST port

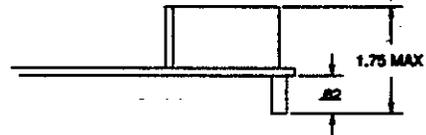
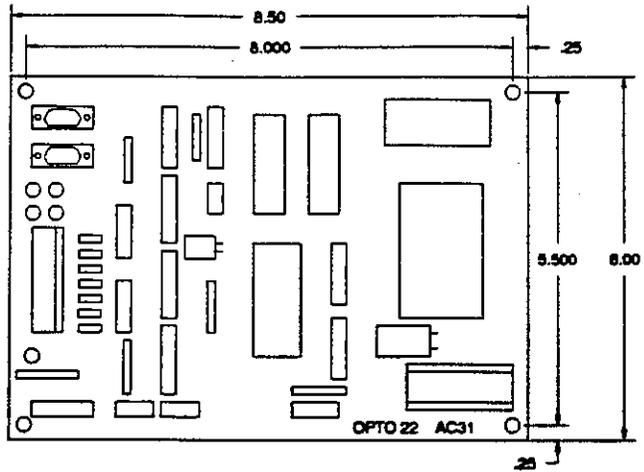
- *On group S and H jumpers, install only one jumper per group for desired BAUD rate.*



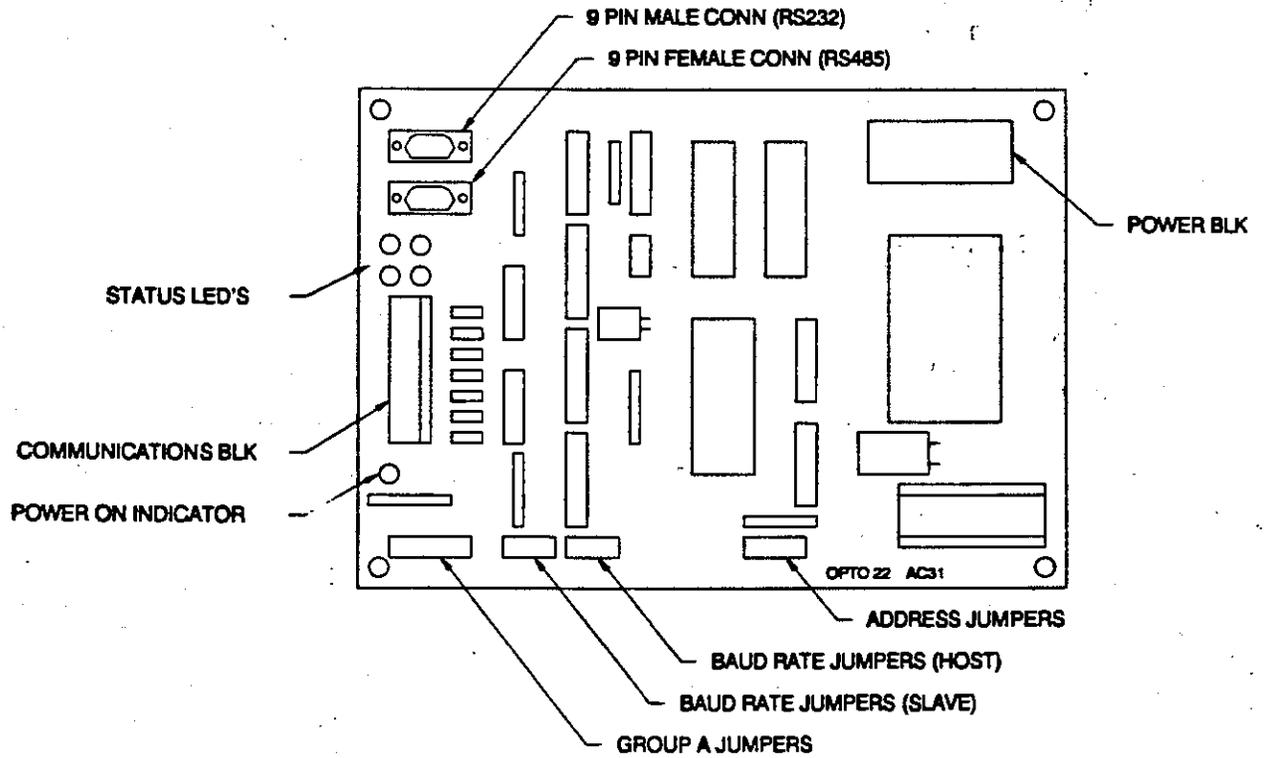
Communications Jumpers Schematic

Address Jumpers

There are eight jumper positions (1 byte) labeled from 0 (LSB) to 7 (MSB) that specify an address between 0 and 255. A jumper installed sets that bit position to a 0, a removed jumper sets that bit position to a 1. All jumpers removed would specify an address of 255. All jumpers inserted specify an address of 0.

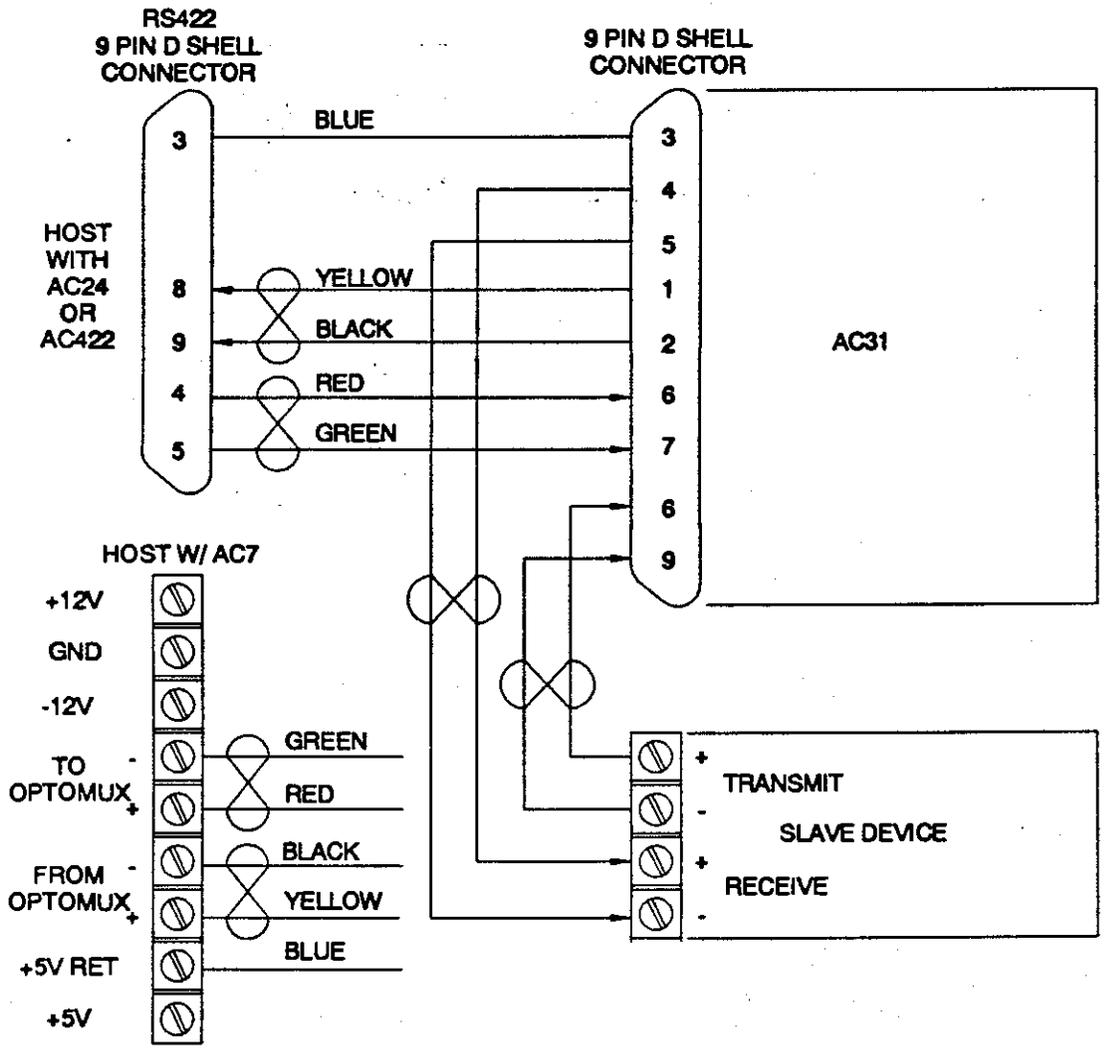


AC31 Dimension Drawing



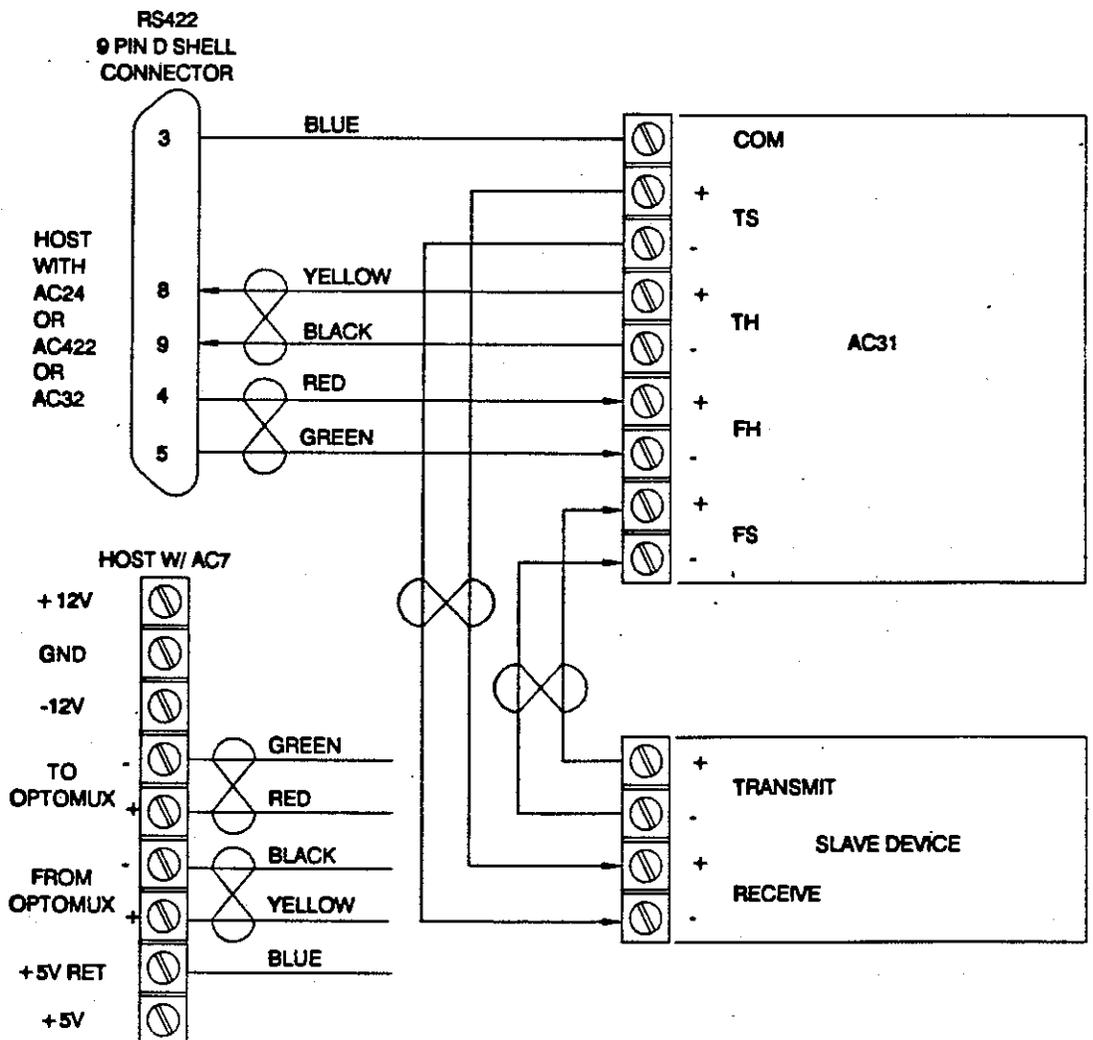
1.3 AC31/SLAVE/HOST WIRING DIAGRAM (Using "D" Shell Connector)

RS485 AC31 "D" SHELL CONNECTOR	DESCRIPTION
1 +TO HOST
2 - TO HOST
3 COMMON
4 + TO SLAVE
5 - TO SLAVE
6 + FROM HOST
7 - FROM HOST
8 + FROM SLAVE
9 - FROM SLAVE



1.4 AC31/SLAVE/HOST WIRING DIAGRAM (Using Screw Terminals)

AC31 SCREW TERMINAL	DESCRIPTION
COM	COMMON
+TS	+ TO SLAVE
-TS	- TO SLAVE
+TH	+ TO HOST
-TH	- TO HOST
+FS	+ FROM SLAVE
-FS	- FROM SLAVE
+FH	+ FROM HOST
-FH	- FROM HOST



1.5 AC31/SLAVE RS232 CONNECTOR WIRING

The AC31 is supplied with an RS-232 port offering RTS/CTS handshake capability. The RS-232 connector uses a male nine pin D shell, its pinout is:

RS232 AC31 "D" SHELL CONNECTOR	DESCRIPTION
1	No Connection
2	Transmit (TX)
3	Receive (RX)
4	Request-To-Send (RTS)
5	Clear-To-Send (CTS)
6	No connection
7	Ground (GND)
8	No Connection
9	Data Terminal Ready (DTR)

1.6 COMMAND STRUCTURE

The AC31 command set is modeled after the OPTOMUX protocol. Each command is composed of 6 parts:

- 1) start of command character
- 2) AC31 address
- 3) AC31 command character
- 4) data field
- 5) command checksum
- 6) end of command character

Start of Command Character

The AC31 uses the same start of command character that is used by OPTOMUX, >. This character precedes all commands that are sent to the AC31.

AC31 Address

The AC31 address is composed of two ASCII hex characters. This gives the AC31 the address range of 0 to 255 (00 to FF hex). Both characters have to be upper case.

Command Character

The command character is a single ASCII letter. It may be upper or lower case, depending upon the command being sent to the AC31.

Data Field

The data field is a string of 0 to 250 characters. All characters in the data field MUST be printable ASCII characters (characters between 20 hex and 7e hex). If non-printable characters need to be passed, they will use escape sequences similar to the ones used in the C language. If the > character is part of the data to be sent to the AC31, it must also be sent as an escape sequence.

Checksum

The command checksum is composed of two ASCII hex characters. It is the sum of the address characters, command letter and data field characters modulo 256 (remainder of sum divided by 256). The start of command character > is NOT part of the checksum.

In the command

>FFACDcr

the checksum CD is the sum of F, F and A (46 Hex + 46 Hex + 41 Hex = CD Hex) modulo 100 Hex (256 decimal). The remainder of CD Hex divided by 100 Hex is CD Hex.

End of Command Character

The end of command character for all AC31 messages is the carriage return (decimal 13).

Example

The following example sends the message "Hello world" terminated by a carriage return to the AC31 addressed at 123:

```
>7BHello world\rD6cr
```

The address, a hex 7B, is a decimal 123.

The command letter S is a print string command.

The data field is Hello world\r

The checksum is calculated by:

<u>Char.</u>	<u>ASCII</u>	
7	37	first address character
B	42	second address character
S	53	command letter
H	48	first data character
e	65	second data character
l	6C	.
l	6C	.
o	6F	.
	20	.
w	77	.
o	6F	.
r	72	.
l	6C	.
d	64	.
\	5C	.
r	+72	last data character

	5D6 Hex	

Now to convert 5D6 to a two character checksum, you perform a modulo 256 upon it. 5D6 modulo 100 (100 hex is 256 decimal) is a D6. This is effectively the same as ANDing 5D6 with a 0FF Hex.

1.7 DATA STRUCTURE

The structure of the data field is modeled after the string formats used in the C language. All printable characters, except for `>`, can be part of the data field. `>` and Non-printable characters are expressed as escape sequences.

The `\` character is used to indicate the start of an escape sequence. Some often used control characters (carriage return, line feed, tab, etc.) are set up as a `\` character and a single lower case letter. Other non-printable characters are expressed as either an octal constant or a hex constant. Examine the table below for the supported escape sequences:

<code>\r</code>	-	carriage return
<code>\n</code>	-	line feed
<code>\t</code>	-	tab
<code>\b</code>	-	backspace
<code>\\</code>	-	backslash character
<code>\f</code>	-	formfeed
<code>\xd</code>	-	hex constant (<i>d</i> is a single hex character)
<code>\xdd</code>	-	hex constant (<i>dd</i> is two hex characters)
<code>\d</code>	-	octal constant (<i>d</i> is a single octal character)
<code>\dd</code>	-	octal constant (<i>dd</i> is two octal characters)
<code>\ddd</code>	-	octal constant (<i>ddd</i> is three octal characters)

The hex and octal escape sequences can be used to represent any of the ASCII character set. For example, a line feed can be expressed as `\n`, `\xA`, `\x0A`, `\12` or a `\012`.

If the message you are sending to the slave contains the character `>`, it must be represented as a hex or an octal constant (`\76` or `\x3E`).

1.8 RETURN MESSAGE STRUCTURE

There are two types of messages that can be returned by the AC31, an acknowledgment message and a no acknowledgment message. An acknowledgment message indicates that the command was performed correctly and the no acknowledgment message indicates that there was some problem with the command.

Acknowledgment Messages

The acknowledgment message is composed of 4 part:

- 1) start of acknowledgment character
- 2) data field
- 3) message checksum
- 4) end of message character

Start of Acknowledgment Character

The AC31 uses the same start of acknowledgment character that is used by OPTOMUX, **A**. This character precedes all messages that are returned to the host.

Data Field

The data field is a string of 0 to 250 characters. All characters returned by the AC31 will be printable ASCII characters (characters between 20 hex and 7e hex). If non-printable characters need to be returned, they will be returned as escape sequences similar to the ones used in the C language. If the **>** character is part of the data to be sent to the host, it must also be sent as an escape sequence.

Checksum

The command checksum is compose of two ASCII hex characters. It is the sum of the data field characters modulo 256 (remainder of sum divided by 256). The start of acknowledgment character **A** is NOT part of the checksum.

In the return message

AAmiga lives22cr

the checksum **22** is the sum of the characters in the string "Amiga lives" ($41 + 6D + 69 + 67 + 61 + 20 + 6C + 69 + 76 + 65 + 73 = 422$) modulo 100 (256 decimal). The remainder of 422 divided by 100 is 22.

End of Acknowledgment Character

The end of command character for all AC31 messages is the carriage return (decimal 13).

No Acknowledgment Message

The no acknowledgment message is composed of 3 parts:

- 1) start of no acknowledgment character
- 2) data field
- 3) end of message character

Start of No Acknowledgment Character

The AC31 uses the same start of no acknowledgment character that is used by OPTOMUX, N. This character precedes all no acknowledgment messages that are returned to the host.

Data Field

The data field is composed of two ASCII hex characters, representing an error condition detected by the AC31. Examine Section 1.8 for detailed information on return error codes.

End of No Acknowledgment Character

The end of command character for all AC31 messages is the carriage return (decimal 13).

1.9 ERROR CODES

The AC31 is capable of detecting 6 different error conditions. The errors are:

Power Up Clear Expected	N00
Undefined Commands	N01
Checksum Error	N02
Input Buffer Overrun	N03
Bad Data	N05
Watchdog Time-Out	N06
Turnaround Time-Out	N07
Bad Message Buffer Number	N08
Send Time-Out	N09
Previous Command Send Time-Out	N0A

Power Up Clear Expected

The **Power Up Clear Expected** error is returned by the AC31 if he receives a command other than the **Power Up Clear** as the first command after power has been applied.

Undefined Command

This message is returned to the host when the command letter received is not part of the AC31 command set.

Checksum Error

The **Checksum Error** message is returned to the host when the calculated checksum does not matched the checksum contained in the received message. This message usually indicates a noisy or an improperly terminated 422 link.

Input Buffer Overrun

This return error tells the host that the message that the AC31 received was longer than 255 characters.

Bad Data

The **Bad Data** error code indicates that the data field passed by the host was not formatted properly for the specified command.

Watchdog Time-out

This return error tells the host that the buss has been inactive for too long of a time.

Turnaround Time-out

The **Turnaround Time-out** error tells the host that the slave device is not talking back to the AC31 properly (or not at all).

Bad Message Buffer Number

The message buffer number specified in the command to the AC31 was beyond the range of 00 to FF hex.

Send Time-out

This return error tells the host that the AC31 has been unable to send it's data to the slave device. Check that the clear to send line on the AC31 RS232 port is being enabled properly.

Previous Command Send Time-out

This return error tells the host that the AC31 was unable to send the data specified in the previous command to the save device. Check that the clear to send line on the AC31 RS232 port is being enabled properly and that the turn around time-out value you have selected is long enough.

2.1 QUICK REFERENCE

LIST OF COMMANDS

<u>COMMAND NAME</u>	<u>COMMAND LETTER</u>
Power Up Clear	A
Reset	B
Set Turnaround Delay	C
Retransmit Last Response	D
Set End of Text Character	E
Return Device Type	F
Read Binary String	G
Write Binary String	H
Get Transmit Status	I
Read ASCII String	L
Read Number of ASCII Characters	N
Output a message buffer to Slave	O
Set Slave Port Configuration	P
Enable Control S/Control Q Protocol	Q
Disable Control S/Control Q Protocol	q
Return Number of Characters Waiting	R
Write ASCII String	S
Write ASCII String and Wait for Response	s
Set Turnaround Time-out	T
Write String to Message Buffer	W
Read a Message Buffer	X

LIST OF ERRORS

<u>ERROR DESCRIPTION</u>	<u>RETURN ERRORS</u>
Power Up Clear Expected	N00
Undefined Commands	N01
Checksum Error	N02
Input Buffer Overrun	N03
Bad Data	N05
Watchdog Time-out	N06
Turnaround Time-out	N07
Bad Message Buffer Number	N08
Send Time-out Error	N09
Previous Command Send Time-out Error	N0A

3.1 COMMAND DESCRIPTIONS

Descriptions of all the AC31 commands are included in this section. The description of each command is formatted as follows:

Purpose:

Tells briefly what the command does.

Format:

Shows the basic structure of the command.

Remarks:

Describes in detail what the command does and how it should be used.

Example:

Shows actual BASIC program segments that demonstrate the use of the command. This manual used Microsoft BASIC as implemented on the IBM PC for its examples.

POWER UP CLEAR

Purpose:

Prevents the AC31 from returning a Power Up Clear Expected error in response to the first instruction following application of power.

Format:

A

Remarks:

This command only functions if it is the first command sent after a power up. If any other command is sent first, a N00 is returned by the AC31. After the N00 error has been returned, the Power Up Clear command does not have to be sent; the AC31 will be ready to execute commands normally.

This command has no effect on the AC31 operation or setup. The Power Up Clear Expected error message provides an indication to the host that there has been a power failure, and that the AC31 has been reset to the power-up configuration.

Example:

Send a Power Up Clear command to the AC31 addressed as 255:

```
1030 PRINT #3,">FFACD";CHR$(13);           'send power up clear command
1040 GOSUB 5000                               'get response from AC31
1050 IF RESPONSE$ <> "A" THEN GOSUB 6000     'if not an Ack., do error routine
```

RESET

Purpose:

This command cause the AC31 to go to its power up state.

Format:

B

Remarks:

The only difference to the AC31 between a Reset Command and a power up conditions is that after a Reset Command, a Power Up Clear command does not have to be sent to the AC31.

The default conditions for the AC31 are:

- All input buffers are empty
- Turnaround delay is 0 seconds.
- Turnaround time-out is 3 seconds.
- End of text character is a carriage return.
- The Control S/control Q protocol is disabled.

Example:

Send a Reset command to the AC31 addressed as 127:

```
1030 PRINT #3,">7FBBF";CHR$(13);           'send reset command
1040 GOSUB 5000                               'get response from AC31
1050 IF RESPONSE$ < > "A" THEN GOSUB 6000    'if not an Ack., do error routine
```

SET TURNAROUND DELAY

Purpose:

Tells the AC31 to wait for a specified time before responding to commands sent from the host.

Format:

C[data]

Remarks:

The data passed by the host is from 0 to 4 ASCII hex characters. The value passed is a time from 0 to 655.35 seconds. The turnaround time delay value is .01 times the value passed by the host. If the host passes a 123 (decimal) to the AC31, the delay will be 1.23 seconds.

Example:

Set the turnaround delay value to .57 seconds on the AC31 addressed as 23.

```
1030 PRINT #3,">17C3917";CHR$(13);  
1040 GOSUB 5000  
1050 IF RESPONSE$ <> "A" THEN GOSUB 6000
```

```
'set turnaround delay to .57 sec.  
'get response from AC31  
'if not an Ack., do error routine
```

RETRANSMIT LAST RESPONSE

Purpose:

Tells the AC31 resend it's last response.

Format:

D

Remarks:

When the AC31 receives this command, it will retransmit it's last response to the host. If there is an error in this command, the previous response will be lost.

Example:

Tell the AC31 addressed as 1 to retransmit it's last message.

1030 PRINT #3,">01DA5";CHR\$(13);

1040 GOSUB 5000

1050 IF RESPONSE\$ < > "A" THEN GOSUB 6000

'tell AC31 to retransmit last message

'get response from AC31

'if not an Ack., do error routine

SET END OF TEXT CHARACTER

Purpose:

Tells the AC31 the end of text character for the slave device.

Format:

E[data]

Remarks:

The data passed by the host is from 0 to 2 ASCII hex characters. The value will now be the termination character for strings received from the slave device. If the host passed a 10 decimal, the end of text character for strings received from the slave is now a line feed. Default is *cr* after power up.

Example:

Set the end of text character to an ASCII ETX (3) on the AC31 addressed as 245.

```
1030 PRINT #3,">F5E3F3";CHR$(13);  
1040 GOSUB 5000  
1050 IF RESPONSE$ <> "A" THEN GOSUB 6000
```

```
'set eot character to a 3  
'get response from AC31  
'if not an Ack., do error routine
```

RETURN DEVICE TYPE

Purpose:

Instructs the AC31 to return it's identification number.

Format:

F

Remarks:

When this command is received, the AC31 will return its device number to the host computer. The AC31 will return 2 ASCII hex characters 1F.

Example:

Read the identity of the board at address 200.

```
1030 PRINT #3,">C8FC1";CHR$(13);           'read device type
1040 GOSUB 5000                             'get response from AC31
1050 IF LEFT$(RESPONSE$,1) < > "A" THEN GOSUB 6000 'if not Ack., do error routine
1060 IF MID$(RESPONSE$,1,2) = "1F" THEN GOTO 2300 'if AC31, go around
```

READ BINARY STRING

Purpose:

Tells the AC31 to return the specified number of binary characters from the slave device.

Format:

G[data]

Remarks:

This command is used for reading data from devices that use binary protocols. The data passed to the AC31 from the host is 0 to 2 ASCII hex characters representing the number of binary characters to read from the slave device.

Each character that the AC31 gets from the slave device is converted to 2 ASCII hex characters before being transmitted to the host computer. A request for more than 125 characters will return an error to the host computer.

Example:

Get 40 binary characters from the AC31 addressed as 0.

```
1030 PRINT #3,">00G2811";CHR$(13);           'read 40 binary chars. from slave
1040 GOSUB 5000                               'get response from AC31
1050 IF LEFT$(RESPONSE$,1) < >"A" THEN GOSUB 6000 'if not Ack., do error routine
```

WRITE BINARY STRING

Purpose:

Send a string of binary characters to the slave device.

Format:

H[data]

Remarks:

The AC31 sends a string of binary characters to the slave device. The data from the host is composed of 2 ASCII hex characters for each binary character to be sent to the slave. If an odd number of data characters are sent by the host, the AC31 will send an error message to the host.

Example:

Send the binary string FRED to the AC31 addressed as 4.

```
1030 PRINT #3,">04H465245444E";CHR$(13);           'send binary string to AC31
1040 GOSUB 5000                                       'get response from AC31
1050 IF LEFT$(RESPONSE$,1) <> "A" THEN GOSUB 6000 'If not Ack., do error routine
```

GET TRANSMIT STATUS

Purpose:

To check the transmit status of the AC31 slave channel.

Format:

I

Remarks:

When this command is received, the AC31 will return either a 1 if it is ready to transmit, or a 0 if it is busy and not ready to transmit. The AC31 checks the state of the ready line by looking at the status register of the UART.

Example:

Read the status of the board at address 200. Continue to loop until board is ready.

```
1030 PRINT #3,"> C8IC4";CHR$(13);           'read status
1040 GOSUB 5000                               'get response from AC31
1050 IF LEFT$(RESPONSE$,1) < > "A" THEN GOSUB 6000 'if not Ack., do error routine
1060 IF MID$(RESPONSE$,1,1) = "0" THEN GOTO 1030 'NOT ready, loop again
```

READ ASCII STRING

Purpose:

Read an ASCII string from the slave, terminated by the current end of text character.

Format:

L

Remarks:

This command reads a line of ASCII characters from the slave, terminated by the current end of text character. If a complete line is not available, an error is returned to the host. Nonprintable characters are returned as hex constants (uses the same escape sequences as the C language uses for its strings).

Example:

Read a string from the AC31 addressed as 42.

```
1030 PRINT #3,">2ALBF";CHR$(13);           'read an ASCII string from the AC31
1040 GOSUB 5000                             'get response from AC31
1050 IF LEFT$(RESPONSE$,1) < >"A" THEN GOSUB 6000 'if not Ack., do error routine
```

READ NUMBER OF ASCII CHARACTERS

Purpose:

Read the specified number of ASCII characters from the slave.

Format:

N[data]

Remarks:

This command receives a specified number of characters from the AC31. If the current end of text character is received before the specified number of characters are received, only the characters up to the end of text character are returned to the host.

The data from the host is composed of 0 to 2 ASCII hex characters, representing the number of characters to read from the slave. If more than 250 characters are requested, an error code is returned to the host computer.

Example:

Read a string of 35 characters from the AC31 addressed as 142.

```
1030 PRINT #3,"> 8EN2330";CHR$(13);           'read an ASCII string of 35 chars.
1040 GOSUB 5000                                 'get response from AC31
1050 IF LEFT$(RESPONSE$,1) < > "A" THEN GOSUB 6000 'if not Ack., do error routine
```

OUTPUT MESSAGE BUFFER TO SLAVE

Purpose:

Write the specified message buffer to the slave device.

Format:

O[data]

Remarks:

This command writes the specified message buffer to the slave device. If the buffer does not contain a message, nothing will be sent to the slave.

The data from the host is composed of 2 ASCII hex characters, representing the message buffer to use (00 through 3F hex). The AC31 has 256 message buffers, so if the requested buffer number is 256 or larger, an error code is returned to the host computer.

Example:

Send the message in buffer number 37 on board 150 to the slave device.

```
1030 PRINT #3,">9602525";CHR$(13);           'send buffer to slave
1040 GOSUB 5000                               'get response from AC31
1050 IF LEFT$(RESPONSE$,1) < > "A" THEN GOSUB 6000 'if not Ack., do error routine
```

SET DEVICE PORT CONFIGURATION

Purpose:

Allow setting of the slave side data bits, stop bits and parity.

Format:

P[data]

Remarks:

This command allows the user to specify the parity, data bits and stop bits for the UART that talks to the slave device. The data passed from the host is composed of 3 characters. The first character can be any of three letters N, E and O. N stands for no parity, E stands for even parity and O stands for odd parity. The parity character can be upper or lower case. The second character specifies the number of data bits per character. It can be one of two values 7 or 8. The third character specifies the number of stop bits per character. It can be one of two values 1 or 2.

Example:

Set the port configuration to even parity, 7 data bits and 2 stop bits on the AC31 addressed as 99.

```
1030 PRINT #3,">63PE7290";CHR$(13);           'set the parity
1040 GOSUB 5000                                'get response from AC31
1050 IF LEFT$(RESPONSE$,1) < >"A" THEN GOSUB 6000 'if not Ack., do error routine
```

ENABLE CONTROL S/CONTROL Q PROTOCOL

Purpose:

Enable control S/control Q handshaking on the slave port.

Format:

Q

Remarks:

When control s/control q is enabled, the AC31 will check for control s and control q characters from the slave. When a control s is detected, the AC31 will stop transmitting to the slave. When a control q is detected, the AC31 will start transmitting characters to the slave.

Example:

Enable control s/control q protocol on the AC31 addressed as 255.

```
1030 PRINT #3,">FFQ51";CHR$(13);           'enable ^s/^q protocol
1040 GOSUB 5000                               'get response from AC31
1050 IF LEFT$(RESPONSE$,1) <> "A" THEN GOSUB 6000 'if not Ack., do error routine
```

DISABLE CONTROL S/CONTROL Q PROTOCOL

Purpose:

disable control S/control Q handshaking on the slave port.

Format:

q

Remarks:

When control s/control q is disabled, the AC31 ignore control s and control q characters.

Example:

Disable control s/control q protocol on the AC31 addressed as 0.

```
1030 PRINT #3,">00qD1";CHR$(13);           'disable ^s/^q protocol
1040 GOSUB 5000                               'get response from AC31
1050 IF LEFT$(RESPONSE$,1) < > "A" THEN GOSUB 6000 'if not Ack., do error routine
```

RETURN NUMBER OF CHARACTERS WAITING

Purpose:

Returns the number of characters in the slave's interrupt buffer.

Format:

R

Remarks:

This command returns the number of characters that are in the slave's interrupt buffer. Up to 16384 characters can be in the slave buffer.

The data returned to the host will be composed of 4 ASCII hex characters with a value of 0000 to 4000 hex.

Example:

Read the number of characters waiting in the slave buffer of the AC31 addressed as 201.

```
1030 PRINT #3,"> C9RCE";CHR$(13);           'read the # of chars. waiting
1040 GOSUB 5000                               'get response from AC31
1050 IF LEFT$(RESPONSE$,1) < >"A" THEN GOSUB 6000 'if not Ack., do error routine
```

WRITE ASCII STRING

Purpose:

Send an ASCII string to the slave device.

Format:

S[data]

Remarks:

This command sends a string to the slave. The AC31 does not wait for a response from the slave, it will send an acknowledgment back to the host after the last character is sent.

The data passed by the host is composed of printable ASCII characters. Any nonprintable characters to be sent to the slave are expressed as escape sequences. The escape sequence format is the same as used in the C language.

Example:

Write the string "This Bud's for you!" with a carriage return appended to the end to the AC31 addressed as 4.

```
1030 PRINT #3,"> 04SThis Bud's for you!\rF7";CHR$(13); 'send string
1040 GOSUB 5000 'get response from AC31
1050 IF LEFT$(RESPONSE$,1) < >"A" THEN GOSUB 6000 'if not Ack., do error routine
```

WRITE ASCII STRING AND WAIT FOR RESPONSE

Purpose:

Send an ASCII string to the slave and wait for a return message.

Format:

s[data]

Remarks:

This command sends a string to the slave. The AC31 will wait for a response from the slave. If no response is received before the AC31 times out, an error message will be sent to the host, else the AC31 will return the data received from the slave.

The data passed to and from the host is composed of printable ASCII characters. Any nonprintable characters to be sent to the slave are expressed as escape sequences. The escape sequence format is the same as used in the C language.

Example:

Write the string "Anybody home? " to the AC31 addressed as 144.

```
.  
1030 PRINT #3,">90sAnybody home? DA";CHR$(13); 'send string  
1040 GOSUB 5000 'get response from AC31  
1050 IF LEFT$(RESPONSE$,1) < >"A" THEN GOSUB 6000 'if not Ack., do error routine  
.  
.
```

SET TURNAROUND TIME-OUT

Purpose:

This command sets the time that the AC31 will wait before it decides that the slave is not responding.

Format:

T[data]

Remarks:

This command allows the user to define how long the AC31 waits for a response from the slave before sending an error message back to the host. The data is passed as 0 to 4 ASCII hex characters. A time-out value from 0 to 655.35 (passed as 0 to FFFF hex) seconds can be selected.

The resolution of the AC31 clock is 10 milliseconds. If a value of 10 (0A hex) is passed, the turnaround time-out will be .1 seconds.

Example:

Set the turnaround time value to 1.6 seconds on the AC31 addressed as 7.

```
1030 PRINT #3,">07TA02C";CHR$(13);           'set turnaround time-out to 1.6 sec.
1040 GOSUB 5000                               'get response from AC31
1050 IF LEFT$(RESPONSE$,1) < > "A" THEN GOSUB 6000 'if not an Ack., do error routine
..
```

WRITE A MESSAGE TO A MESSAGE BUFFER

Purpose:

Put a message into a message buffer.

Format:

W[data]

Remarks:

The AC31 contains 256 message buffers, each holding up to 32 characters. The message buffers are allocated consecutively in memory, so it is possible to have messages larger than 32 characters. Messages of more than 32 characters, will use more than one buffer. A message of 32 characters or less will take up 1 buffer, between 33 and 64 characters will take up 2 consecutive buffers, between 65 and 96 characters will take up 3 consecutive buffers, etc. The only limitation on the message length is the AC31 limitation of a maximum command length of 256 characters.

The data from the host is composed of 2 to 249 ASCII characters. The first 2 characters are ASCII hex, representing the message buffer to use (00 through FF hex). The remaining characters are the message to be stored in the specified buffer.

Example 1:

Put a message into buffer number 7 on board 100.

```
.  
. 1030 PRINT #3,">64W07This program is coded in C4D";CHR$(13);  
1040 GOSUB 5000 'get response from AC31  
1050 IF LEFT$(RESPONSE$,1) < >"A" THEN GOSUB 6000 'if not Ack., do error routine  
.  
.
```

Example 2:

Put a message into buffers number 17 and 18 on board 100.

```
.  
. 1030 PRINT #3,">64W11This message will take up two buffers in the AC3149";CHR$(13);  
1040 GOSUB 5000 'get response from AC31  
1050 IF LEFT$(RESPONSE$,1) < >"A" THEN GOSUB 6000 'if not Ack., do error routine  
.  
.
```

READ MESSAGE BUFFER

Purpose:

Tells the AC31 to return the specified buffer contents to the host computer

Format:

X[data]

Remarks:

This command is used for reading a message buffer from the AC31. The data passed to the AC31 from the host is 0 to 2 ASCII hex characters representing the message buffer to read.

Example:

Read message buffer number 40 from AC31 number 0.

```
1030 PRINT #3,">00X2822";CHR$(13);           'read the specified buffer
1040 GOSUB 5000                               'get response from AC31
1050 IF LEFT$(RESPONSE$,1) < > "A" THEN GOSUB 6000 'If not Ack., do error routine
```

APPENDIX A - BASIC Subroutines

This appendix contains 3 utility subroutines, written in BASIC, useful for building, sending and receiving AC31 messages.

UTILITY ROUTINES

This routine calculates the checksum of the string contained in **ANSWER\$**, and returns the result in **CHECKSUM\$**.

```
10000 '
10010 '
10020 ' Calculate checksum
10030 '
10040 ' This subroutine calculates the checksum for the string ANSWER$
10050 ' and puts it into the string CHECKSUM$
10060 '
10070 '
10080 CHECKSUM% = 0 'initialize checksum
10090 FOR X% = 1 TO LEN( ANSWER$ ) 'set up loop
10100 CHECKSUM% = CHECKSUM% + ASC(MID$(ANSWER$,X%,1)) 'add in 1 character
10110 NEXT
10120 '
10130 CHECKSUM$ = RIGHT$("00" + HEX$(CHECKSUM%),2)'make checksum% into ASCII
10140 RETURN
10150 '
```

This routine verifies the checksum of the message contained in **RESPONSE\$**, and returns the status in **ERRORS%**

```
11000 '
11010 '
11020 ' Verify Checksum
11030 '
11040 ' This subroutine verifies that the checksum for the string
11050 ' RESPONSE$ is good, if good, ERRORS% will contain a 0, if the
11060 ' is bad, ERRORS% will equal -31 (OPTOWARE driver checksum error)
11070 '
11080 '
11090 CHECKSUM% = 0
11100 TEMP% = VAL("&H" + RIGHT$(RESPONSE$,2)) 'get received checksum
11110 FOR X% = 2 TO LEN( RESPONSE$ ) - 2 'set up loop
11120 CHECKSUM% = CHECKSUM% + ASC(MID$(RESPONSE$,X%,1)) 'add in 1 character
11130 NEXT
11140 '
11150 IF CHECKSUM% = TEMP% THEN CHECK.FLAG% = -1 ELSE CHECK.FLAG% = 0 'set flag
11160 RETURN
11170 '
```

This routine receives a message from the AC31. The message is stored in **RESPONSE\$**, and **ERRORS%** will contain the status of the operation.

```
12000 '  
12010 '  
12020 ' Get AC31 response  
12030 '  
12040 ' This routine gets a response for the AC31. The received  
12050 ' message will be stored in RESPONSE$. If a complete response is  
12060 ' not received (message terminated by a carriage return), the  
12070 ' variable ERRORS% will contain a -29.  
12080 '  
12090 '  
12100 RESPONSE$ = ""           'initialize input string  
12110 TEMP = TIMER           'set up for time-out  
12120 A$ = ""                'initialize input buffer  
12130 ERRORS% = 0            'clear the error variable  
12140 WHILE ASC(RIGHT$(A$,1)) < > 13 'wait for a return  
12150 IF LOC(1) = 0 THEN GOTO 12180 'if no characters, go around  
12160 A$ = INPUT$(LOC(1),1)   'get some characters  
12170 RESPONSE$ = RESPONSE$ + A$ 'append character to response  
12180 IF (TIMER-TEMP) < 2 THEN GOTO 12210 'see if we're still in time  
12190 ERRORS% = -29         'on a time-out, set error code  
12200 A$ = CHR$(13)         'clean exit of while/wend loop  
12210 WEND  
12220 RETURN  
12230 '
```

APPENDIX B - ASCII Character Set

<u>Decimal</u>	<u>Hex</u>	<u>Character</u>	<u>Decimal</u>	<u>Hex</u>	<u>Character</u>
0	00	NUL	32	20	Space
1	01	SOH	33	21	!
2	02	STX	34	22	"
3	03	ETX	35	23	#
4	04	EOT	36	24	\$
5	05	ENQ	37	25	%
6	06	ACK	38	26	&
7	07	BEL	39	27	'
8	08	BS	40	28	(
9	09	HT	41	29)
10	0A	LF	42	2A	*
11	0B	VT	43	2B	+
12	0C	FF	44	2C	,
13	0D	CR	45	2D	-
14	0E	SO	46	2E	.
15	0F	SI	47	2F	/
16	10	DLE	48	30	0
17	11	DC1	49	31	1
18	12	DC2	50	32	2
19	13	DC3	51	33	3
20	14	DC4	52	34	4
21	15	NAK	53	35	5
22	16	SYN	54	36	6
23	17	ETB	55	37	7
24	18	CAN	56	38	8
25	19	EM	57	39	9
26	1A	SUB	58	3A	:
27	1B	ESC	59	3B	;
28	1C	FS	60	3C	<
29	1D	GS	61	3D	=
30	1E	RS	62	3E	>
31	1F	US	63	3F	?

(Ascii Character Set, Continued)

<u>Decimal</u>	<u>Hex</u>	<u>Character</u>	<u>Decimal</u>	<u>Hex</u>	<u>Character</u>
64	40	@	96	60	'
65	41	A	97	61	a
66	42	B	98	62	b
67	43	C	99	63	c
68	44	D	100	64	d
69	45	E	101	65	e
70	46	F	102	66	f
71	47	G	103	67	g
72	48	H	104	68	h
73	49	I	105	69	i
74	4A	J	106	6A	j
75	4B	K	107	6B	k
76	4C	L	108	6C	l
77	4D	M	109	6D	m
78	4E	N	110	6E	n
79	4F	O	111	6F	o
80	50	P	112	70	p
81	51	Q	113	71	q
82	52	R	114	72	r
83	53	S	115	73	s
84	54	T	116	74	t
85	55	U	117	75	u
86	56	V	118	76	v
87	57	W	119	77	w
88	58	X	120	78	x
89	59	Y	121	79	y
90	5A	Z	122	7A	z
91	5B	[123	7B	{
92	5C	\	124	7C	
93	5D]	125	7D	~
94	5E	^	126	7E	
95	5F	_	127	7F	DEL

OPTO 22

**15461 Springdale St., Huntington Beach, CA 92649
(714) 891-5861 (800) 854-8851 FAX: (714) 893-0867**