6

*Direct*NET Host Communications Programs

In This Chapter. . . .

Why do you need a communications program?

The Master Initiates Requests

Since ***Direct***NET is a master / slave network, the master station must initiate requests for network data transfers. If you're using a host as the master station, you will need to use a communications program written with the ***Direct***NET protocol.

***Direct*NET Programs**

The communications program used with a hosted network is more complex than the simple RLL instructions used with the other configurations, but the concept is the same. The host is the ***Direct***NET master and must use a ***Direct***NET protocol communications program to initiate all network requests to read or write data. These communication programs can be written in many different languages, such as BASIC, C, etc. and must include the appropriate ***Direct***NET commands.

Here's an example of a ***Direct***NET program. (This is just part of the program.)

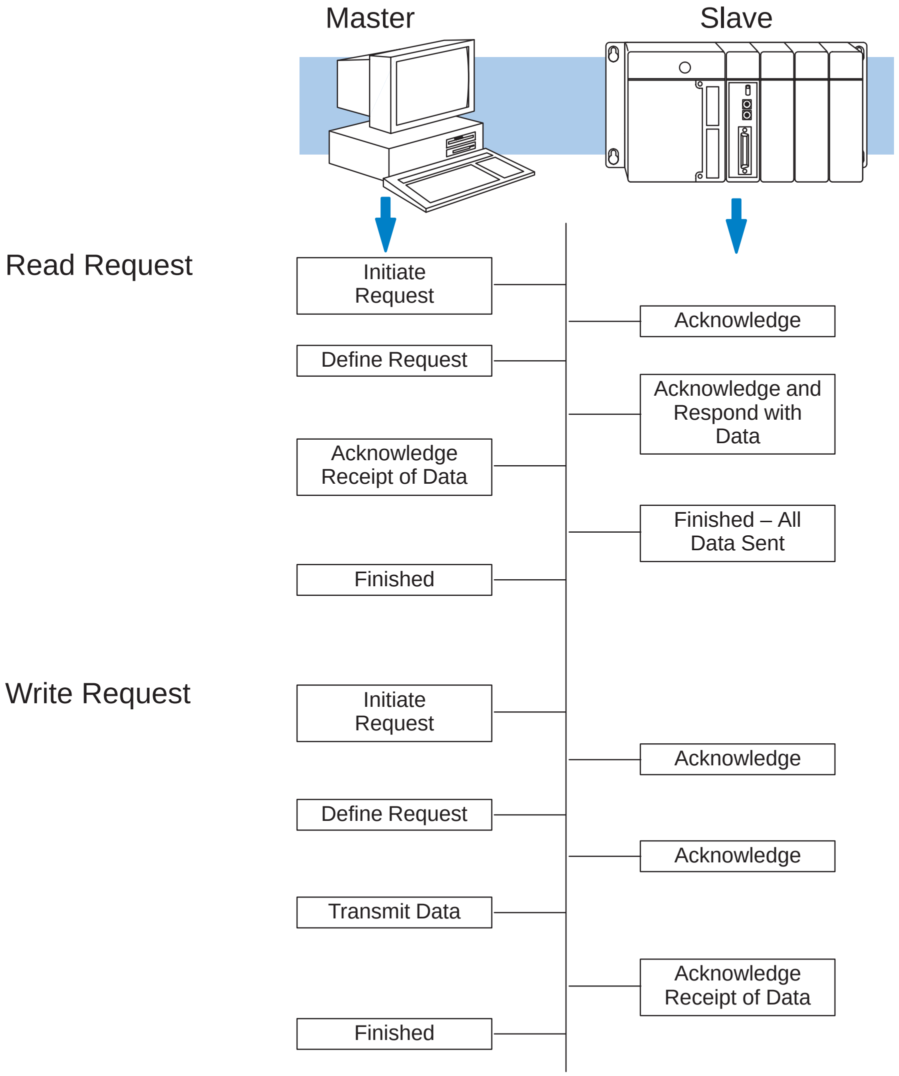
DirectNET Program in BASIC

```
10    REM Program to read X0–X7 from a DL405 PLC
20    REM
30    REM Define all variables
40    REM
50    REM Change the slave address in HEX at line 60 if required.
60    SLAVEADDRESS=&H1
70    DATATYPE$=CHR$(&H32)
80    DATAADDR$=CHR$(&H30)+CHR$(&H31)+CHR$(&H30)+CHR$(&H31)
90    COMPLETEBLK$=CHR$(&H30)+CHR$(&H30)
100   PARTBLK$=CHR$(&H30)+CHR$(&H32)
110   MASTERADDR$=CHR$(&H30)+CHR$(&H30)
120   NORMAL$=CHR$(&H4E)
130   SLAVEADDR$=HEX$(SLAVEADDRESS)
140   IF LEN(SLAVEADDR$)<2 THEN SLAVEADDR$="0"+SLAVEADDR$
150   OFFSETADDR$=CHR$(&H20+SLAVEADDRESS)
|
|
|
```

NOTE: This manual does not show you how to build communications programs that manage the data storage and communications ports. You should check the documentation that came with your programming software to determine the appropriate techniques to solve these requirements.

The following diagram shows the general structure of the communications. The program must:

- identify the slave station.
- indicate the type and amount of data to transfer.
- manage the communications between the master and slave.



The remainder of this chapter discusses the individual elements of ***Direct***NET protocol programs.

Modes of Operation

Transmission Bytes

***Direct*NET can transfer a maximum of 65,791 bytes (256 blocks 256 bytes each + an additional 255 bytes) in a single request. The actual amount of system information that is transferred depends on the mode of operation.

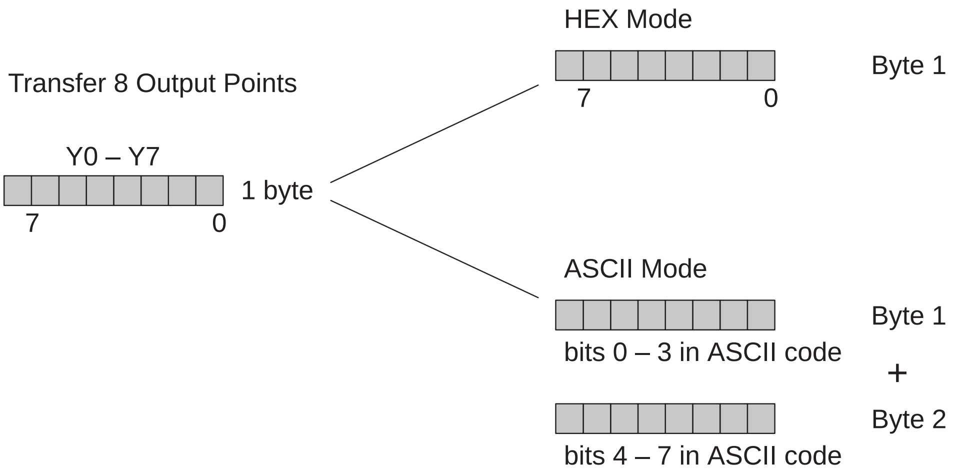
HEX or ASCII Mode

There are two modes used with the ***Direct*NET protocol, HEX or ASCII. You must choose the mode of operation before you write the program. The major difference is in the way the data is represented in the data packet. ASCII mode requires twice as many bytes to transfer data. There are also minor differences in the command structure which affect the way the enquiries and headers operate.

NOTE: This only applies to a Host master, since the DCM has the capability to detect the data transfer mode and adjust for the additional number of bytes to be used when transferring in ASCII mode.

If you're transferring small amounts of data, or if the data is not being used to control system timing, then it's generally easier to use ASCII mode for ***Direct*NET programs. You should use HEX mode if you're transferring large amounts of data and you need the fastest possible communication time.

The following diagram shows the difference between HEX and ASCII modes.



Data Type Byte Requirements

In HEX mode, the number of bytes transferred is equal to the number of bytes for the selected data type. ASCII mode requires twice as many bytes to transfer the same data. Here's a listing of the data types and their corresponding byte requirements.

DL205/405 Data Type	Description	Bits per unit	Number of bytes	
			HEX	ASCII
31	V memory T / C current value	16 16	2 2	4 4
32	Inputs (X, GX, SP)	8	1	2
33	Outputs (Y, C, Stage, T/C bits)	8	1	2
39	Diagnostic Status	8	1	2

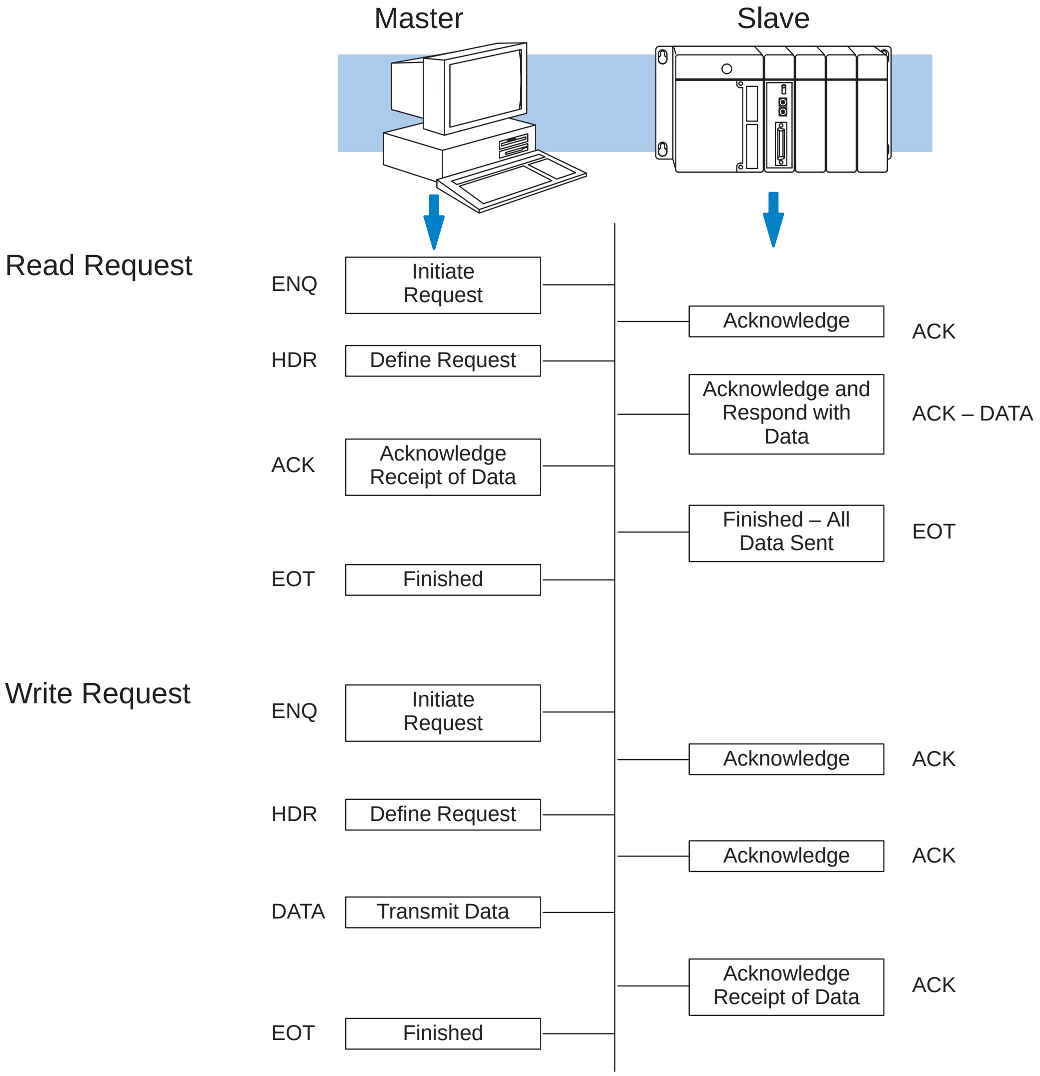
DL305 Data Type	Description	Bits per unit	Number of bytes	
			HEX	ASCII
31	Data registers T / C accumulator	8 16	1 2	2 4
33	I/O, internal relays, shift register bits, T/C bits, stage bits	1	1	2
39	Diagnostic Status (5 word R/W)	16	10	20

Protocol Components

All ***Direct***NET program read and write requests use the following protocol components.

- Enquiry (ENQ) – initiates a request (from the master) with the slave stations.
- Header (HDR) – defines the operation as a read or write, the slave station address, and the type and amount of data to be transferred.
- Data (DATA) – the actual data that is being transferred.
- Acknowledge (ACK) – verifies communication is working correctly.
- End of Transmission (EOT) – indicates the communication is finished.

The following diagram shows how the protocol components are used with read and write requests.



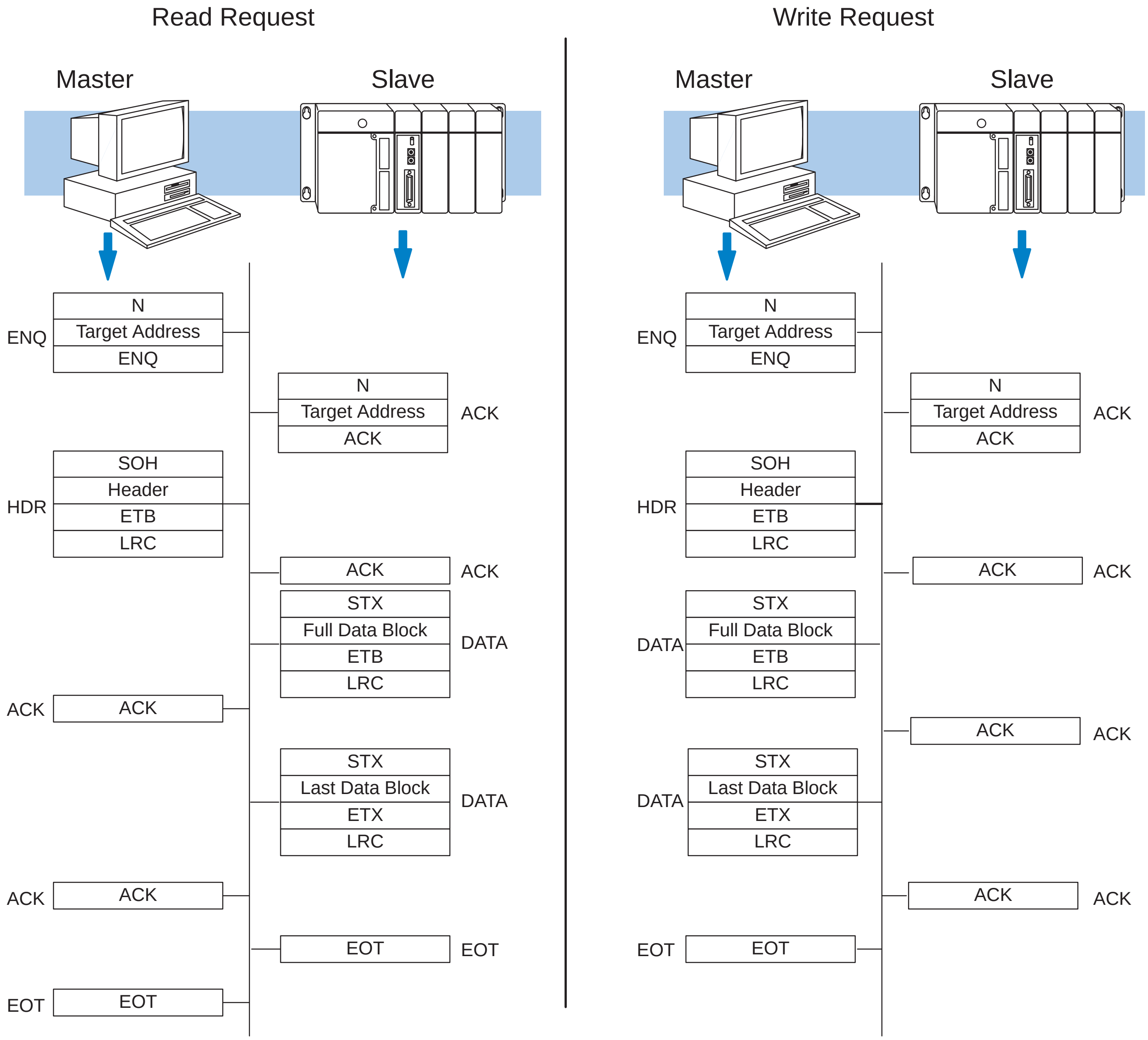
Controlling the Communications

Control Codes

All read or write requests use ASCII control codes and a Longitudinal Redundancy Check (LRC) to manage the communications between the master and slave. The control codes identify the beginning and ending of the protocol components such as, enquiry, acknowledge, etc. The LRC is used to ensure the data was transmitted and received correctly.

Symbol	HEX ASCII Code	Description
ENQ	05	Enquiry – initiate request
ACK	06	Acknowledge – the communication was received without error
NAK	15	Negative Acknowledge – there was a problem with the communication
SOH	01	Start of Header – beginning of header
ETB	17	End of Transmission Block – end of intermediate block
STX	02	Start of Text – beginning of data block
ETX	03	End of Text – End of last data block
EOT	04	End of Transmission – the request is complete.

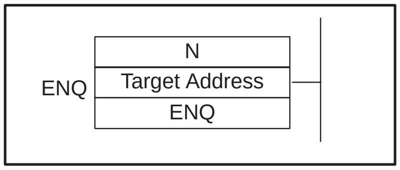
The following diagram shows how these control codes are combined with the protocol components to build the ***Direct*NET format for read and write requests. Note, the slave components are automatically generated as a response to the requests from the master station. Your custom ***Direct*NET program must generate the protocol components for the master station.



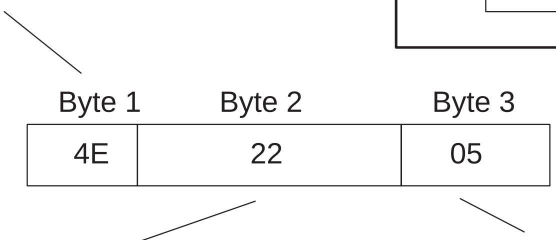
Initiating the Request

Enquiry ENQ

The Enquiry is a three character message that initiates the request with the appropriate slave station. The message always begins with 4E (“N”), which means normal enquiry sequence. The second character contains the offset station address, which is the station address plus HEX 20. The last character is the ASCII control code for ENQ.



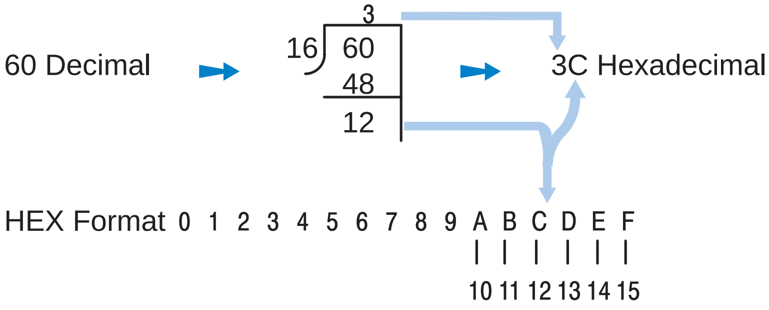
Hexadecimal ASCII code for “N”



Offset slave address.
Example –
Address HEX 2 + HEX 20 = HEX 22

ASCII control code for ENQ

Note: Slave addresses for the DL430, DL440, DL340 and the DCU have been set in decimal. It will be necessary for you to convert the address from decimal to the Hexadecimal equivalent before adding the HEX 20 offset.



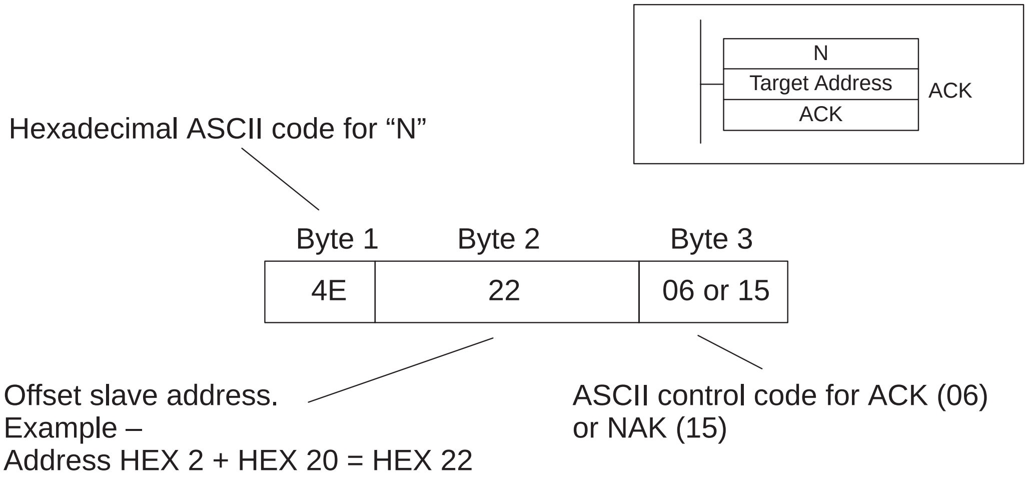
Acknowledging the Request

Acknowledge ACK – NAK

The three character acknowledge commands are used by both the master and slave stations to indicate the status of the communication. An ACK is used if the information was transmitted (or received) without any problems. If there are problems, Not Acknowledge (NAK) is used.

A NAK will also be returned from the slave if something is incorrect in the header or data packet. This could be incorrect byte boundaries, an invalid address, etc. If the master receives a NAK response, it can either try to re-transmit the data, or it can terminate the request and try again.

The first two characters are the same as the Enquiry sequence. The third character is the control code for an ACK or NAK.



Hexadecimal ASCII code for "N"

Byte 1	Byte 2	Byte 3
4E	22	06 or 15

Offset slave address.
Example –
Address HEX 2 + HEX 20 = HEX 22

ASCII control code for ACK (06) or NAK (15)

Delayed Response to an Enquiry

When text is being transmitted over the network, there may be character combinations that are identical to an enquiry sequence. Network slave stations would interpret the character sequence as an enquiry even though it was actually data being sent to an identified slave.

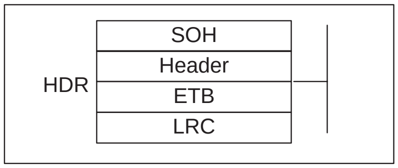
A delay has been implemented to automatically eliminate this possibility. The delay occurs between the receipt of an enquiry from the master and the acknowledgment response from the slave. When the slave recognizes an enquiry sequence an internal timer (with the time preset to the amount of time to transmit 2 characters) is started. The slave ignores the enquiry if another character is received before the timeout period has elapsed.

Defining the Request

Header – HDR

The header is a 17-byte (18-byte for ASCII transmissions) message that defines the operation. It is sent by the master station and specifies the following.

- type of operation (read or write)
- type of data being transferred
- data address
- number of complete data blocks
- number of bytes in the last data block



ASCII Coded Representation Example

Byte:	1	2, 3	4	5	6, 7	8, 9	10, 11	12, 13	14, 15	16	17(18)	
	01	3034	30	31	3431	3031	3031	3930	3031	17	08 3038	Hex ASCII
	SOH	Target Slave	Read or Write	Data Type	Starting Address MSB	Starting Address LSB	Number Com-plete Blocks	Bytes in Last Block	Mas-ter ID	ETB	LRC	

**Byte 1:
Start of Header**

The first byte in the header is the ASCII control code (01) that indicates this is the beginning of a header.

**Bytes 2 & 3:
Target Slave
Address**

The second and third bytes of the header indicate which slave station will be used. This is the normal slave station address (in HEX ASCII code) that you assigned during the network setup. This *is not* the offset slave station address, (with 20 HEX added to the address), that is used with the enquiry sequence. For example, a slave station with address 04 would be 3034 in ASCII code.

The table below shows how to decode the HEX/ASCII slave address. Remember if the slave address is in HEX it will be necessary to translate the address to decimal before decoding the HEX ASCII address. In the example below a slave station has a network address of 04 decimal and the equivalent HEX ASCII code is 3034.

Decimal or ASCII Station Address
04

ASCII Character	0	1	2	3	4	5	6	7	A	B	C	D	E	F
HEX Code	30	31	32	33	34	35	36	37	41	42	43	44	45	46

3034
Target Slave Address
in HEX ASCII Code

Byte 4: Read or Write

Byte 4 indicates whether the operation is a read or write request. A value of HEX ASCII 30 is read, HEX ASCII 38 is write.

Byte 5: Data Type

This byte identifies the type of memory to be accessed. Appendices D–F provide a complete listing of the data types and memory references for product families.

Bytes 6 & 7: Starting Address MSB

The address is the starting point for the data transfer. The data is transferred from this point forward. For example, to transfer the first 32 X input points from a DL405 PLC the starting address would be V40400. The request would actually obtain V40400 and V40401 since there are 16 points per V-memory location.

Bytes 6 and 7 define the most significant byte of the ASCII coded memory address. For example, the reference address for V40400 is 4101. This is obtained by converting the octal number to hexadecimal and adding 1, the most significant byte of this value is then decoded into HEX ASCII. The most significant byte would be HEX 41 or HEX ASCII 3431. Appendices D–F provide complete references for the addresses used in the various ***Direct***LOGIC™ PLC families.

Bytes 8 & 9: Starting Address LSB

These bytes define the least significant byte of the address obtained in the step above. So to continue on with our example of reading the first 32 inputs at memory location V40400 from a DL405 PLC, the reference value for this location would be 4101. The least significant byte of the reference value (01) would be decoded to 3031 in HEX ASCII.

Bytes 10 & 11: Complete Data Blocks

This is the coded number of complete data blocks that should be transferred. ***Direct***NET can transfer 256 bytes in a single data block. Take the number of bytes and divide by 256 to obtain the number of complete data blocks. This results in a valid range of 00 – FF HEX, or 3030 – 4646 HEX ASCII. If you're not transferring at least 256 bytes, then this field should be HEX 00 or HEX ASCII 3030.

For example, if you are transferring 200 V-memory locations, you would have the following: 200 x 2 bytes per location = 400 bytes. 400 bytes / 256 per block =1 complete block, with 144 bytes remaining. For one (01) complete data block, the value entered for this field would be the HEX ASCII code of 3031.

Bytes 12 & 13: Partial Data Block

This is the HEX ASCII coded number of bytes in the last data block. If you did not have a an even number of complete data blocks, then you had some remaining bytes. Enter the number of remaining bytes here. The valid range is 00 – FF HEX, or 3030 – 4646 HEX ASCII.

For example above with 144 bytes (or 90 bytes in HEX) the value entered in this field would be the HEX ASCII code of 3930.

Bytes 14 & 15: Master Station ID

This is the master station address. Since the master station should be address 0 or 1, this field is always HEX ASCII 3030 or 3031 for addresses 0 and 1 respectively.

Byte 16: End Transmission

Byte 16 always contains the HEX ASCII code for End of Transmission Block. This field always contains HEX ASCII17.

Byte 17: Longitudinal Redundancy Check

This is a checksum that is used to verify the communications were received without any errors. This is calculated by taking the exclusive OR of the bytes between the start of header (SOH) and the end of transmission (ETB) bytes (bytes 2 – 15). If you're using ASCII format, then this actually takes two bytes which makes the header an 18 byte message. The LRC is explained in more detail later in this chapter.

Transferring Data

Data Blocks

The data blocks contain the actual data that is being transferred between the master and slave station. ***Direct***NET transfers data in full blocks of 256 8-bit bytes, or partial blocks of less than 256 8-bit bytes. The 256 byte limit does not include control characters that signal the end of the data. To determine the number of full blocks, divide the number of bytes by 256. The remainder is the number of bytes in the partial data block.

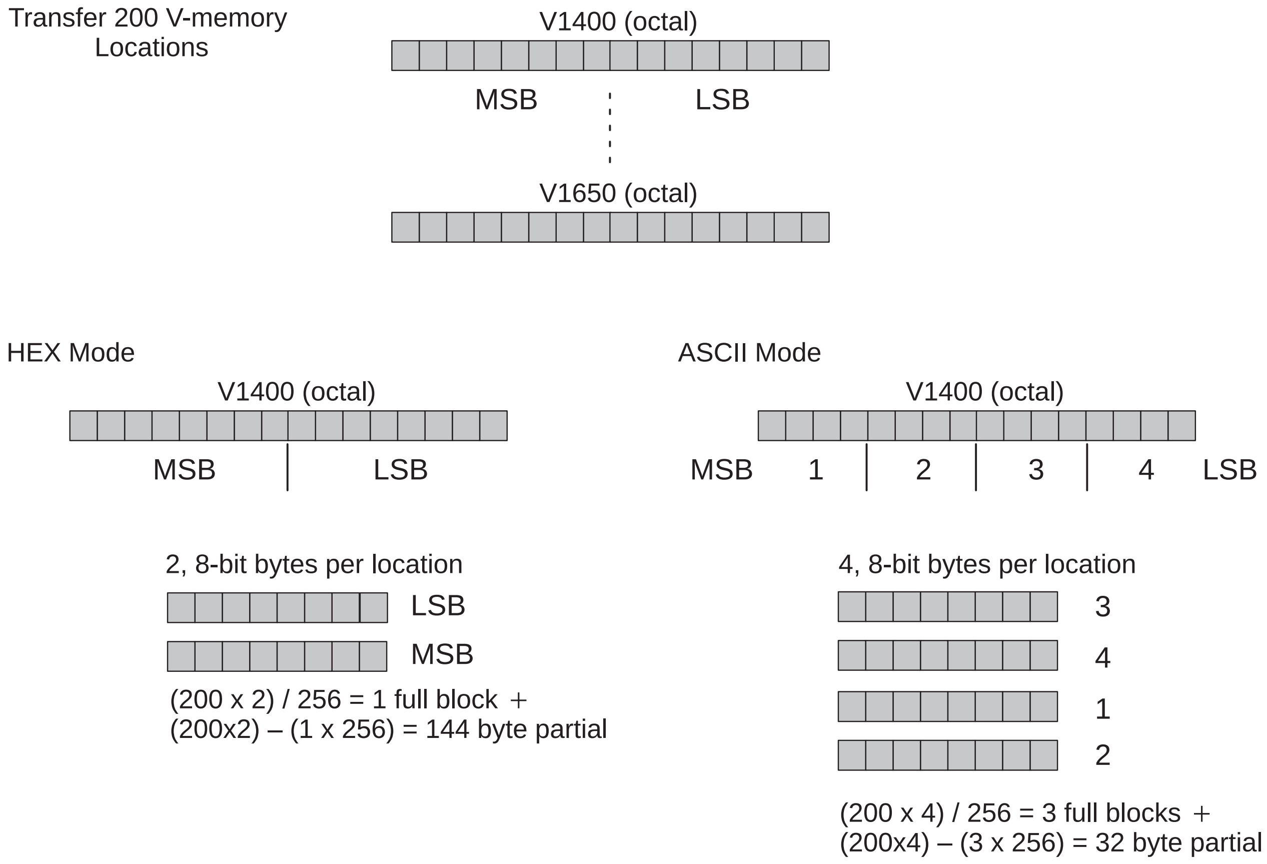
Since ASCII mode requires twice as many bytes for the data, you can transfer more information per request with HEX mode.

Transmission Sequence

Since the data is transmitted in bytes, it is important to understand how the original value is separated during transmission. ***Direct***NET uses a simple byte swapping process where the least significant byte is transferred first. In ASCII mode, the original data is split into 4-bit units and then converted into 8-bit bytes.

Memory types that only use 1 byte are also treated in the same manner. For example, a 1-byte memory type would yield 1 byte in HEX mode, but two bytes in ASCII mode (4 bits converted into 2, 8-bit bytes).

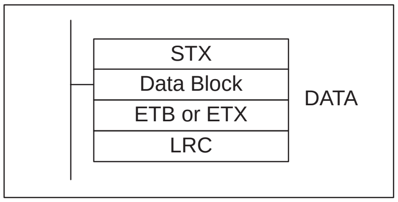
The following diagram shows the differences between HEX and ASCII modes.

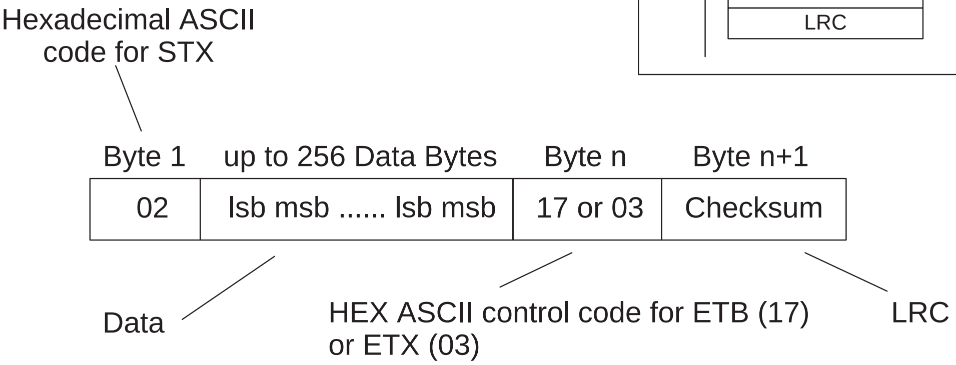

Transfer 200 V-memory Locations
V1400 (octal)
MSB
LSB
V1650 (octal)
HEX Mode
V1400 (octal)
MSB
LSB
2, 8-bit bytes per location
LSB
MSB
(200 x 2) / 256 = 1 full block +
(200x2) – (1 x 256) = 144 byte partial
ASCII Mode
V1400 (octal)
MSB
1
2
3
4
LSB
4, 8-bit bytes per location
3
4
1
2
(200 x 4) / 256 = 3 full blocks +
(200x4) – (3 x 256) = 32 byte partial

Start of Text, End of Block, End of Text

The HEX ASCII control codes that indicate the beginning and end of data blocks are used to manage the data transfer. Start of Text (STX) indicates the beginning of a data block. If there are several blocks, all but the last block will terminate with the End of Block (ETB) code. The last block always ends with End of Text (ETX). All transfers also include an LRC checksum. (For a data block, the checksum is the exclusive OR of all bytes between the STX and ETB/ETX characters. The LRC is discussed in more detail later.)

The following diagram shows the communication sequence.





Calculating the Header Checksum

Longitudinal Redundancy Check

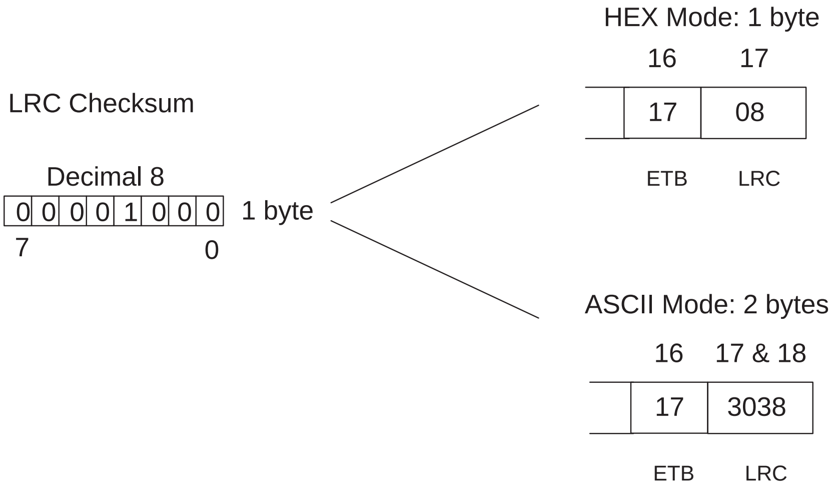
The LRC yields a checksum which is used to verify the communications are being received without errors. For a header, this checksum is calculated by taking the exclusive OR of all bytes between the Start of Header and End of Transmission (ETB). For a data block, the checksum is the exclusive OR of all bytes between the STX and ETB/ETX characters. To take the exclusive OR, just convert the HEX values to binary and then examine the bits. For each bit position, an even number of '1's results in a checksum value of 0. An odd number of '1's results in a checksum value of 1. Here's an LRC calculation example based on the values used in the discussion of the header.

HEX ASCII Coded Representation

Byte:	1	2, 3	4	5	6, 7	8, 9	10, 11	12, 13	14, 15	16	17(18)	
	01	3034	30	31	3431	3031	3031	3930	3031	17	08 3038	HEX ASCII
	SOH	Target Slave	Read or Write	Data Type	Data Address MSB	Data Address LSB	Number Complete Blocks	Bytes in Last Block	Master ID	ETB	LRC	

Byte		HEX ASCII Value	Binary Representation							
			B7 128	B6 64	B5 32	B4 16	B3 8	B2 4	B1 2	B0 1
Target slave:	(byte 2)	30	0	0	1	1	0	0	0	0
	(byte 3)	34	0	0	1	1	0	1	0	0
Read or write:	(byte 4)	30	0	0	1	1	0	0	0	0
Data Type:	(byte 5)	31	0	0	1	1	0	0	0	1
Data address MSB:	(byte 6)	34	0	0	1	1	0	1	0	0
	(byte 7)	31	0	0	1	1	0	0	0	1
Data address LSB:	(byte 8)	30	0	0	1	1	0	0	0	0
	(byte 9)	31	0	0	1	1	0	0	0	1
Complete blocks:	(byte 10)	30	0	0	1	1	0	0	0	0
	(byte 11)	31	0	0	1	1	0	0	0	1
Bytes in last block:	(byte 12)	39	0	0	1	1	1	0	0	1
	(byte 13)	30	0	0	1	1	0	0	0	0
Master address:(byte 14)		30	0	0	1	1	0	0	0	0
	(byte 15)	31	0	0	1	1	0	0	0	1
Total Number of "1s"			0	0	14	14	1	2	0	6
Even (E) or Odd (O)			E	E	E	E	O	E	E	E
Exclusive OR Results:			0	0	0	0	1	0	0	0
Hexadecimal Value			0				8			
HEX ASCII Code			30				38			

From the table the checksum value is decimal 8. The checksum is contained in byte 17 of the header, but the actual value that is included depends on which mode of operation you are using. In HEX mode, this would be HEX 08. If you're using ASCII mode, then the value would be 3038 and the LRC now requires two bytes (17 and 18). The following diagram shows the differences in a HEX or ASCII mode LRC.



Calculating the Data LRC

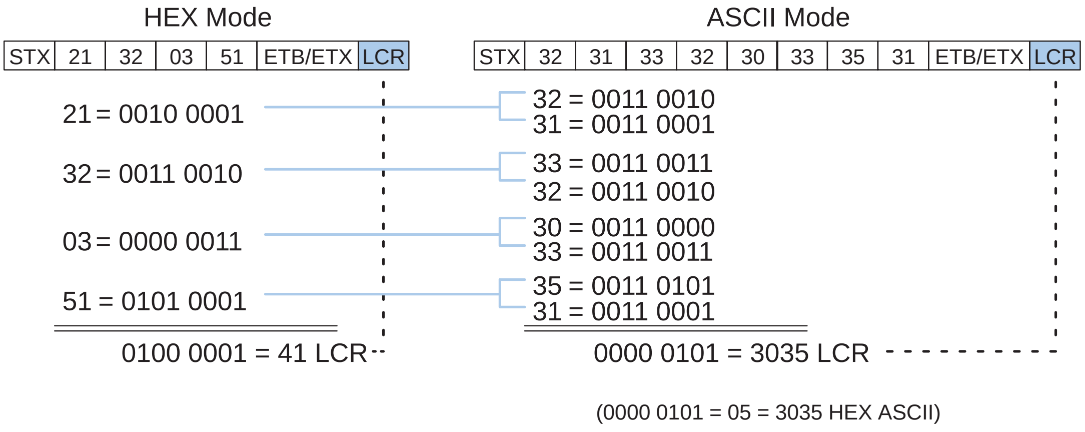
You should always calculate the LRC when either writing data to a slave or reading data from a slave. Note, during a read command slave stations will calculate their own LRC to be verified by the receiving device. However, you must calculate the value for LRC verification.

The LRC is included in the header and data transmissions. For a data block, the checksum is the exclusive OR of all bytes between the STX and ETB/ETX characters. (The example program in Appendix C shows how to do this in a BASIC program.)

NOTE: You only have to verify the checksum when you are creating your own ***Direct***NET communications programs. If the master is a ***Direct***LOGIC™ PLC with a communications interface, RLL instructions are used for the communication program and checksum verification is done automatically.

LRC Example for HEX and ASCII Transfers

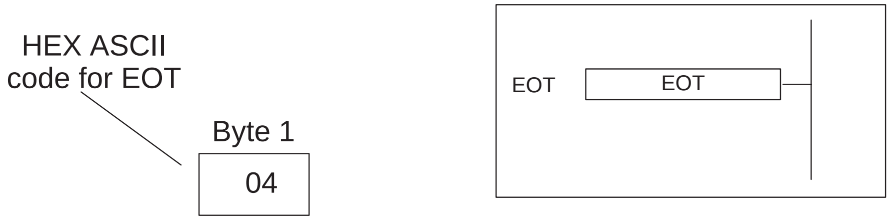
This example shows how to calculate the LRC for the same data being transferred in either HEX or ASCII mode.



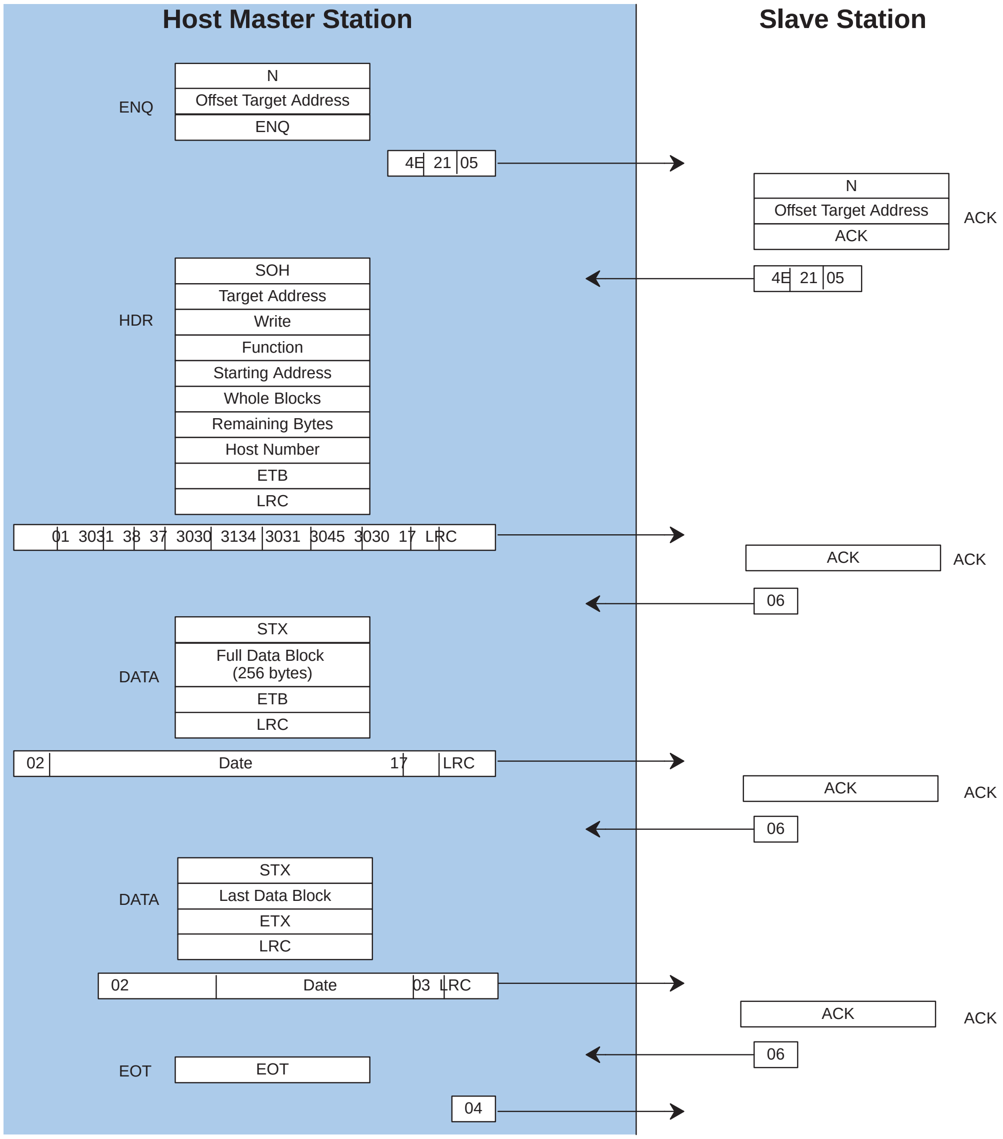
Ending the Request

End of Transmission – ETB

When the last data block has been transferred and acknowledged, the End of Transmission (EOT) character is sent. The master station must always end the communication by sending an EOT (HEX ASCII 04). The following diagram shows the EOT format.



Master to Slave Data Transfer Summary Sheet



Timing Considerations

Timeouts

The network communications generally operate very quickly and without problems. However, as with all things, problems can occur. Timeouts occur when either the master or slave does not receive a response to a communication within a certain period of time. There are two timeout possibilities.

- Slave timeout – this occurs when the slave does not respond within a specified time. When the slave times out, you must send an EOT from the host to terminate the communication
- Master timeout – this occurs when the slave station does not receive the complete communication from the master within a specified time. The slave station will send an EOT to signal that the communications have been terminated. The master must also send an EOT back to the slave to acknowledge the termination. (This allows the next communication to begin.)

The following tables provide the maximum times that ***Direct***LOGIC™ products will wait for a communication before entering a timeout condition.

DL405 Timeouts

Communication Segment	within (ms)
Master sends ENQ → Slave sends ACK	800
Master receives ACK → sends Header	800
Slave receives Header → sends ACK/NAK	2000
(Destination – write is slave, read is master) Master receives ACK → Data is transferred Destination receives Data → sends ACK/NAK Source receives ACK/NAK → sends EOT	 20000 20000 800

DL205 Timeouts

Communication Segment	within (ms)
Master sends ENQ → Slave sends ACK	800
Master receives ACK → sends Header	800
Slave receives Header → sends ACK/NAK	2000
(Destination – write is slave, read is PLC) Master receives ACK → Data is transferred Destination receives Data → sends ACK/NAK Source receives ACK/NAK → sends EOT	 20000 20000 800

DL305 Timeouts

Communication Segment	within (ms)
Master sends ENQ → Slave sends ACK	800
Master receives ACK → sends Header	800
Slave receives SOH → waits for Header LRC	
300 baud	2670
1200 baud	670
9600 baud	670
19.2K baud	670
Slave receives Header → sends ACK/NAK	2000
(Destination – write is slave, read is PLC)	
Master receives ACK → Data is transferred	20000
Destination receives Data → sends ACK/NAK	
300 baud	33340
1200 baud	8340
9600 baud	8340
19.2K baud	8340
Source receives ACK/NAK → sends EOT	800

Managing Timeouts

All communications finish with an EOT being sent from the master station. Even if the slave station sends an EOT to signal that it has aborted the communication, the master still must send an EOT to enable the slave to accept a new enquiry.

If you are reading information from a slave and the LRC calculated in your program does not match the slave station LRC, you do not have to abort the communication with an EOT. Instead, send a NAK which will signal the slave to re-transmit the data. If you send an EOT you must restart the entire request.

Communication Retries

The slave stations send NAKs to signal the master to try sending header or data packets again. The master can either re-transmit a maximum of three times, or, send an EOT to restart the request.

Delays

Each portion of the communication requires a delay to allow the PLC to process the information. If you send data without allowing for the delay, the communication may be ignored (which causes a timeout), or the parity check will fail (which causes a NAK response). The following table provides delay times for the ***Direct*LOGIC™** products.

Information to Process	Delay (ms)
ACK of data packets, headers	1
All other ACKs, headers, EOTs	1

NOTE: The communication interfaces have delay switches that increase this delay time. If those delays are selected, add the appropriate times to the figures shown here.

What part of the manual should you use next?

Start the Network

Once you’ve created the communications program, you can start the network. Chapter 7 provides information concerning network operation and troubleshooting. Appendix C provides an example of a hosted network.