SIXNET Universal Protocol

A Universal way of reliably interfacing a distributed SIXNET system to any computer using any operating system.

Table of Contents

Section 1 _____

Introduction

The SIXNET Universal Protocol is the "Universal" way to talk to all SIXNET products, including VersaTRAK RTUs, and SIXTRAK, RemoteTRAK and EtherTRAK I/O. It is also backward compatible with the older IOMUX and VERSAMUX RTUs.

Each Universal Protocol message is packeted, acknowledged, and error checked to make communications reliable over fault prone media such as radio links and telephone lines. Each message also includes return (source) addressing to reply to messages over multi-mastered networks such as RS485 party-lines and Ethernet or other networked systems. This means that multiple computers can address the same I/O simultaneously and that peer-to-peer communications between stations can occur concurrent with I/O polling from master stations.

This flexible communications protocol exchanges all types of I/O. In addition to the common discrete and analog values, you can also move floating point numbers, long integers, character strings, and even user defined data structures.

This manual describes the format of Universal Protocol messages and makes suggestions to help you implement an I/O driver in your own application.

A sample DOS program is supplied to help you get started. In addition, an easy to use Windows program is provided to act as an I/O simulator. Any computer can temporarily become a mock SIXNET station to test your driver. More details on the tools provided on the ST-UDR Driver Development Support Disk are provided in section 4.

Important Note: If you are using a Windows-based system, The SIXNET Control Room software handles I/O updating to SIXNET stations. Interface to your Windows applications using the DLL calls documented in the Control Room on-line help or use the DDE server included in the Control Room.
You don't need to write a driver or read this manual if you use the SIXNET Control Room.

Section 2 _____

Universal Protocol Messages

2.1 List of Commands

These are the publicly supported Universal Protocol commands. Usage and syntax information appears later in this manual.

Name	Description	Value
NOP	Do nothing command	0
ACK	Acknowledge valid message	1
NAK	Not acknowledge -- Bad message received	2
VERS	Mutually exchange version information	3
NIO	Report number of I/O type specified	4
GETD	Get (read) a block of discrete values	10
GETB	Get (read) a block of bytes (characters)	11
GETA	Get (read) a block of analog values	12
GETS	Get a string terminated by 0 (null char)	13
PUTD	Send (write) a block of discrete values	14
PUTB	Send (write) a block of bytes (characters)	15
PUTA	Send (write) a block of analog values	16
SETD	Turn on individual discrete bits	17
CLRD	Turn off individual discrete bits	18
SCLOCK	Set station's real-time clock/software timer	24
GCLOCK	Read station's real-time clock/software timer	25
IOXCHG	Update all I/O in a distributed I/O module	32
Users	Values reserved for user defined commands	96-127

2.2 General Message Format

Universal Protocol messages are sent in this general format:

<lead><length><dest><source><session><sequence><command>...data...[crc]

Throughout this manual '< >' brackets indicate a one byte value, '[ ]' brackets indicate a two byte value, which is sent in big-endian order, most significant byte first, unless stated otherwise.

2.2.1 Lead Character -- 1 byte

The first message character defines all of the other bytes of the message as either hexadecimal or binary format. Binary format packs eight bits of data into each character of the message and is the more efficient of the two formats. Hexadecimal format puts each data byte into two successive characters as an ASCII '0'-'9',

'A'-'F', or 'a'-'f' for systems that cannot transmit eight bit data bytes. `Fixed CRC' is binary format with a fixed CRC field, and is recommended to save computation time by eliminating the active CRC over media such as Ethernet that already provide CRC message checking.

Sending messages in Hex Format merely changes the encoding of the message bytes. The Hex Format message is exactly the Binary Format message with the lead character changed and each byte encoded as 2 hexadecimal characters. The value of the Length Character does not change to reflect the doubled number of characters sent.

	Lead Character
Binary Format	)
Hex Format	]
Fixed CRC	**}**

2.2.2 Length Character -- 1 byte

This field indicates the total length of the message including all characters after the message length field up to and including the CRC bytes at the end of the message. The maximum value is 255. Please note that special considerations exist for extended station addressing. See 2.2.3.1.

2.2.3 Destination Station Address -- 1 or 2 bytes

Specifies the receiving station number.

For station numbers 0 - 127, send a one byte station number.
0 - 127: Simple 1 byte station number.
255: Any station respond (don't use this bradcast address on a multi-station network)
Note: One byte addresses 128 - 254 are illegal.

For station numbers 128 - 15999, send two bytes:
1st byte: 80h + most significant six bits
2nd byte: Least significant eight bits
Example: Station number 12345 would be B039h
Note: Addresses greater than 15999 are illegal.

2.2.3.1 Special considerations for two byte addresses:

A. Each of the two bytes are calculated into the CRC.
B. The entire two byte address is counted as only one byte in the message length field.

2.2.4 Source Station Address -- 1 or 2 bytes

Specifies the sending station number to facilitate routing of reply messages and is the same format as the destination station address.

2.2.5 **Session** -- 1 byte

This field allows multiple links to be established between two stations simultaneously. The same session number must be assigned to the corresponding buffers in both stations. Session numbers between the same two stations must be unique. Session number assignments between different station pairs are unrelated and may be duplicated.

0:	Default session number (fixed in most applications)
1 - 127:	User definable (for advanced applications)
128 - 255:	Reserved codes that you must not use

2.2.6 **Sequence Number** -- 1 byte

It is recommended that a sequential number (0-255, cyclical) be sent in each message to insure that messages have not been lost and to match replies to commands. SIXNET stations echo this character in reply messages. Set this field to '0' to ignore this feature.

2.2.7 **Command** -- 1 byte

This numeric value identifies the type of command message being sent. See section 2.1 for a list of command values.

2.2.8 **Data field** -- variable

This field contains I/O data and/or other information and varies for each type of message, and does not exist in all message types. Refer to the individual message descriptions.

The data field may contain addressing information (defined for each message type) and a variable size determined as follows:

<type>	I/O Type number byte. See section 2.4 and 3.4.
[start]	First register address to be transferred.
[num]	The number of registers to be transferred.
D	Discrete messages: Number of bytes = num / 8
A	Analog messages: Number of bytes = num * 2
B	Byte messages: Number of bytes = num

2.2.9 **CRC** -- 2 bytes

An error checking code will be attached to all messages and replies. All characters are included in this check except the lead character, the CRC bytes, and the optional terminator characters. See section 3.1 for details.

2.2.10 Message Terminator Characters -- variable

Optional CR and/or LF characters after Hex Format messages will be echoed by the receiving station after the reply, but are otherwise ignored. This allows a master station needing these characters to terminate a reply to force the responding station to send them at the end of the reply.

2.3 Content of Messages

The specific format for each type of message is given in this section, alphabetically by command name. First, some general notes:

1. [dddd] represents a 2 or 4 byte field. Section 3.3 describes the data order.
2. <dd> represents a one byte field.
3. <'d'> represents an ASCII character.
4. <ddh> represents a hexadecimal number (lower case h)
5. These field descriptions appear in the block I/O move commands:
   <type> I/O type -- 1 byte. See 2.4, 3.3 and 3.4
   [start] first I/O register to update -- 2 bytes
   [num] number of registers to update -- 2 bytes
6. Sample commands are all shown in binary message format.
7. The destination and source bytes for the sample commands have been set to 255 (any station).
8. The session byte has been set to 0 for all examples.
9. The 'sequence' byte in the examples is set to 5 to distinguish it and to serve as a place marker within the message.

ACK Command byte value = 1

This message acknowledges that the command was received and processed. Any data returned by the command is also sent in the ACK message. The format of the ACK message depends on the command it is responding to. See other commands for examples.

CLRD Command byte value = 18

This command is used to turn off (set = 0) only specific discrete values (bits). A 1 (ON) bit in the data field of the message will turn OFF the corresponding discrete value. A 0 (OFF) bit in the data field will have no effect. This command does a bitwise AND of the discretes specified with the ones complement of the data field.

Command: (14 characters + data)
<lead><length><dest><source><session><sequence><CLRD><type>[start][num]...<data>...[crc]
Acknowledge Reply: (14 characters)
<lead><length><dest><source><session><sequence><ACK><type>[start][num][crc]

Example:
This CLRD command sends a mask of 8 discrete outputs of type DEF_OUT starting with Y0. Y1, Y3, Y5 and Y7 will be turned off.

Command: (15 characters)
<')'><13><255><255><0><5><18><1><0><0><8><0><170><93><135>
Acknowledge Reply: (14 characters)
<')'><12><255><255><0><5><1><1><0><0><8><0><38><255>

GCLOCK Command byte value = 25

This command is used to read a software seconds timer. The command is valid with IOMUX, SIXTRAK or VersaTRAK products. If a real-time clock is present on the station, the software seconds timer is loaded with the real-time clock value each time the station is powered up. The software seconds timer is read by the SCIL function time(). IOMUX firmware v5.3 and later responds to this command.

Note: 'time' is a 4 byte unsigned long representing the number of seconds since Jan 1, 1970. The time will be sent in big-endian format, <MSB>...<LSB>.
Example: 5:10 AM July 31, 1991
2896462fh <28h><96h><46h><2fh>

Command: (9 characters)
<lead><length><dest><source><session><sequence><GCLOCK>[crc]
Acknowledge Reply: (13 characters)
<lead><length><dest><source><session><sequence><ACK>[time][crc]

Example:
Request the current value of the software seconds timer on the IOMUX. The 4 byte time value returned represents a time of: Wed Apr 29 05:20:00 1992

Command: (9 characters)
<')'><7><255><255><0><5><25><84><79>
Acknowledge Reply: (13 characters)
<')'><11><255><255><0><5><1><41><254><50><0><75><253>

GETA Command byte value = 12

This command is used to read a block of analog I/O values. Use this command to read inputs or verify (read back) outputs.

Command: (14 characters)
<lead><length><dest><source><session><sequence><GETA><type>[start][num][crc]
Acknowledge Reply: (variable length)
<lead><length><dest><source><session><sequence><ACK><type>[start][num]...[data]...[crc]

Example:
This GETA command asks for 1 analog input (type 0) starting with the first element in the default analog input array. Multiple byte fields (start and num) will be sent in little-endian format. Note that the type field is set to 0 which is the sum of TYPE + LITTLE_END_FORMAT (0 + 0). The slave station will reply with the values of analog input array element 0.

Command: (14 characters)
<')'><12><255><255><0><5><12><0><0><0><1><0><120><117>
Acknowledge Reply: (16 characters)
<')'><14><255><255><0><5><1><0><0><0><1><0><0><0><145><144>

GETB Command byte value = 11

This command is used to read a block of bytes (typically characters or discretes). 8 discrete or 1 character will be read by each byte of the reply message. Use this command to read inputs or characters or verify (read back) outputs. This command is different from GETD in that `start' and `num' specify byte quantities instead of the number of bits (discretes). The data bytes in the reply are sent as stored in the station, they are not affected by big/little-endian selection in the type byte, since if this command is used to retrieve multi-byte values there is no way for the driver to know how to rearrange the bytes.

Command: (14 characters)
<lead><length><dest><source><session><sequence><GETB><type>[start][num][crc]
Acknowledge Reply: (variable length)
<lead><length><dest><source><session><sequence><ACK><type>[start][num]...<data>...[crc]

Example:
This GETB command asks for 1 byte of discrete inputs (type 0) (default input). The start and num fields will be sent in little-endian format. The slave station will reply with the values of discrete inputs 0 thru 7 (1 byte of data).

Command: (14 characters)
<')'><12><255><255><0><5><11><0><0><0><1><0><176><52>
Acknowledge Reply: (15 characters)
<')'><13><255><255><0><5><1><0><0><0><1><0><170><105><79>

GETD Command byte value = 10

This command is used to read a block of discrete I/O values. Use this command to read inputs or verify (read back) outputs. The 'start' and 'num' fields are specified in bit quantities. For IOMUX and Versamux stations only, 'start' and 'num' must be specified on byte boundaries (multiples of eight). SIXTRAK stations allow any values.

Command: (14 characters)
<lead><length><dest><source><session><sequence><GETD><type>[start][num][crc]
Acknowledge Reply: (variable length)
<lead><length><dest><source><session><sequence><ACK><type>[start][num]...<data>...[crc]

Example:

This GETD command asks for 8 discrete inputs (type 0). The multiple start and num fields will be sent in Little-Endian format. The slave station will reply with the values of discrete inputs 0 thru 7 (1 byte of data).

Command: (14 characters)
<')'><12><255><255><0><5><10><0><0><0><8><0><79><12>
Acknowledge Reply: (15 characters)
<')'><13><255><255><0><5><1><0><0><0><8><0><170><247><222>

GETS Command byte value = 13

This command is used to read a string of characters from a character array. This command differs from the GETB command in that `num' (number of bytes to read) is not specified. Instead, a variable length null (0) terminated string is returned. The GETS command has the advantage of only returning the actual string of characters instead of the entire buffer (maximum allowable string length). If the string is greater than 244 characters long it will be truncated. The zero (null termination character) will be returned after the data.

Command: (12 characters)
<lead><length><dest><source><session><sequence><GETS><type>[start][crc]
Acknowledge Reply: (variable length)
<lead><length><dest><source><session><sequence><ACK><type>[start]...<0 to 244 characters>...<0>[crc]

Example:
This GETS command asks a string from the array defined by type 0. The length of the reply depends on the data at the address specified. For this example, the single character '7' is stored at offset 0 in the type 0 array, and is returned by the function.

Command: (12 characters)
<')'><10><255><255><0><5><13><0><0><0><35><250>
Acknowledge Reply: (28 characters)
<')'><12><255><255><0><5><1><0><0><0><'7'><0><153><5>

IOXCHG Command byte value = 32

This special I/O updating command exchanges discrete and analog inputs and outputs in a single command. It is specifically intended to efficiently update all I/O in a RemoteTRAK or EtherTRAK I/O module. By sending and receiving inputs and outputs with a single command, the turnaround time for messages is improved.

This command assumes that all I/O requests start with the first register (X0, AX0, Y0, and AY0) and sequential registers are addressed. All I/O data is sent/returned in lowest to highest register order. Note that the maximum length of a UDR message is 255 bytes, so if the total number of bytes of combined I/O data in this message exceeds that limit, the message is invalid and will be NAK'ed.

The 'DI Count' and 'DO Count' fields are specified in byte quantities. Discrete data is bit-packed eight registers per byte (LSB = lowest and MSB = highest). For these fields, any value from 0 to four must be valid. ('0's may be sent for I/O that is not physically present). Values greater than four may be NAK'ed if the polled station is incapable of processing more I/O. Discrete counts larger than four bytes are legal for any future products which support larger quantities of I/O registers. The Discrete Count fields use six bits (d0 – d5) to specify the number of bytes sent/requested) The d6 bit should be set to "0" to read values (typical for inputs or to read back outputs) and set to "1" to write values (typical for outputs).

The 'AI Count' and 'AO Count' fields are specified in Word (16-bit register) quantities. These fields use six bits (d0 - d5) to specify the number of analog registers sent/requested. The high bit (d7) is used to select between Intel (0 = Little Endian) and Motorola (1 = Big Endian) format for the Analog registers in the data section of the message/reply. The d6 bit is "0" to indicate reading, and "1" to indicate writing. For these fields, any value from 0 to 16 must always be valid. (Zeros may be returned for I/O that is not physically present). Values over sixteen may be NAK'ed if the hardware is incapable of processing more I/O. Analog Counts larger than sixteen registers are legal for any future products which support this.

Note: Only select SIXNET products support this command.

Command: (variable length)

<lead><length><dest><source><session><sequence><IOXCHG><DI Count><DO Count><AI Count><AO Count><Discrete Ins(if written)><Discrete Outs(if written)>[Analog Ins (if written)][Analog Outs (if written)][crc]

Acknowledge Reply: (variable length)

<lead><length><dest><source><session><sequence><ACK><DI Count><DO Count><AI Count><AO Count><Discrete Ins(if read)><Discrete Outs(if read)>[Analog Ins(if read)][Analog Outs(if read)][crc]

Example:

This IOXCHG command sends eight discrete outputs and four analog outputs. It also requests eight discrete inputs and four analog inputs. The sending station address is 1. The responding I/O module's station address is 2. This command will write outputs and read inputs. Analog values are in "Big Endian" format.

Discrete Outputs	Analog Outputs	Discrete Inputs	Analog Inputs
Y0 = OFF	AY0 = $0001	X0 = ON	AX0 = $1000
Y1 = ON	AY1 = $0002	X1 = OFF	AX1 = $2000
Y2 = OFF	AY2 = $0003	X2 = OFF	AX3 = $3000
Y3 = ON	AY4 = $0004	X3 = OFF	AX4 = $4000
Y4 = OFF		X4 = OFF	
Y5 = ON		X5 = OFF	
Y6 = OFF		X6 = OFF	
Y7 = ON		X7 = OFF	

Command: (22 characters)

<')'><20><2><1><192><79><32><1><65><132><196><170><0><1><0><2><0><3><0><4><210><112>

Acknowledge Reply: (22 characters)

<')'><20><1><2><192><79><1><1><65><132><196><1><16><0><32><0><48><0><64><0><230><144>

NAK Command byte value = 2

This message is returned when a properly formatted message is received but the commanded action could not be completed. It is not sent if the CRC check failed. Reasons for NAK replies include unrecognized or unsupported commands, referring to I/O that does not exist, and others.

Command: (9 characters)
<lead><length><dest><source><session><sequence><NAK>[crc]

Example:
This example is the NAK reply that would be given for any of the command examples listed in this manual.

Command: (9 characters)
<')'><7><255><255><0><5><2><247><21>

NIO Command byte value = 4

This command reports the number of defined I/O points of the TYPE specified. For types less than 6 both the discrete and analog count of that TYPE will be reported. If the discrete or analog array of that TYPE is not defined, a 0 will be reported. For types 6 and above the same count is reported for 'num discrete' and 'num analog', and if the type is not defined 0 will be reported.

Command: (10 characters)
<lead><length><dest><source><session><sequence><NIO><type>[crc]
Acknowledge Reply: (14 characters)
<lead><length><dest><source><session><sequence><ACK><type>[num discrete][num analog][crc]

Example:
This command asks for the number of physical I/O (type 4) from the slave station. The slave station replies that it has 256 physical discrete inputs (X) and 96 physical analog inputs (AX).

Command: (10 characters)
<')'><8><255><255><0><5><4><4><244><60>
Acknowledge Reply: (14 characters)
<')'><12><255><255><0><5><1><4><0><1><96><0><176><27>

NOP Command byte value = 0

This command specifies no operation and produces an ACK reply.

Command: (9 characters)
<lead><length><dest><source><session><sequence><NOP>[crc]

Acknowledge Reply: (9 characters)
<lead><length><dest><source><session><sequence><ACK>[crc]

Example:
This NOP command does nothing but produce an ACK reply. The slave station replies with an ACK reply that has no data.

Command: (9 characters)
<')'><7><255><255><0><5><0><215><87>
Acknowledge Reply: (9 characters)
<')'><7><255><255><0><5><1><199><118>

PUTA Command byte value = 16

This command is used to write a block of 16-bit analog I/O values, typically outputs. Although it is possible to write to input locations, it is not generally recommended.

Command: (14 characters + data)
<lead><length><dest><source><session><sequence><PUTA><type>[start][num]...[data]...[crc]
Acknowledge Reply: (14 characters)
<lead><length><dest><source><session><sequence><ACK><type>[start][num][crc]

Example:
This PUTA command writes the value 291 to 1 analog output (type 1).

Command: (16 characters)
<')'><14><255><255><0><5><16><1><0><0><1><0><35><1><16><2>
Acknowledge Reply: (14 characters)
<')'><12><255><255><0><5><1><1><0><0><1><0><156><103>

PUTB Command byte value = 15

This command is used to write a block of discrete I/O values, typically outputs, or characters (bytes). Although it is possible to write to input locations, it is not generally recommended. This command is different from PUTD in that `start' and `num' specify byte quantities instead of the number of bits (discretes).

Command: (14 characters + data)
<lead><length><dest><source><session><sequence><PUTB><type>[start][num]...<data>...[crc]
Acknowledge Reply: (14 characters)

<lead><length><dest><source><session><sequence><ACK><type>[start][num][crc]

Example:
This PUTB command writes 1 discrete output byte (type 1). Multiple byte fields (start and num) will be sent in little-endian format.

Command: (15 characters)
<')'><13><255><255><0><5><15><1><0><0><1><0><170><30><103>
Acknowledge Reply: (14 characters)
<')'><12><255><255><0><5><1><1><0><0><1><0><156><103>

PUTD Command byte value = 14

This command is used to write a block of discrete I/O values, typically outputs. Although it is possible to write to input locations, it is not generally recommended. The 'start' and 'num' fields are specified in bit quantities. For IOMUX and Versamux stations only, 'start' and 'num' must be specified on byte boundaries (multiples of eight). SIXTRAK stations allow any values.

Command: (14 characters + data)
<lead><length><dest><source><session><sequence><PUTD><type>[start][num]...<data>...[crc]
Acknowledge Reply: (14 characters)
<lead><length><dest><source><session><sequence><ACK><type>[start][num][crc]

Example:
This PUTD command writes 8 discrete outputs (type 1). Multiple byte fields (start and num) will be sent in little-endian format.

Command: (15 characters)
<')'><13><255><255><0><5><14><1><0><0><8><0><170><56><151>
Acknowledge Reply: (14 characters)
<')'><12><255><255><0><5><1><1><0><0><8><0><38><255>

SCLOCK Command byte value = 24

This command sets the real-time clock (if present) and the software timer on an SIXNET station. The time is passed as the 'time' field (a 4 byte unsigned long representing the time as the number of seconds since Jan 1, 1970). The ACK response to the SCLOCK command also contains a clock status byte. Clock status codes are described below.

Command: (13 characters)
<lead><length><dest><source><session><sequence><SCLOCK>[time][crc]
Where 'time' is a 4 byte unsigned long representing the number of seconds since Jan 1, 1970. The time must be sent in big-endian format, <MSB>...<LSB>.
Example: 5:10 AM July 31, 1991
2896462fh <28h><96h><46h><2fh>
Acknowledge Reply: (10 characters)
<lead><length><dest><source><session><sequence><ACK><clk_status>[crc]
Where clk_status is a one byte field whose value may be:
For IOMUX/Versamux:
0 - no real time clock (only software timer set)
1 - real time clock on expansion port 1 set
2 - real time clock on expansion port 2 set
3 - real time clock on main board (not set)

4 - clock set by user slave task
For SIXTRAK:
0 - no real time clock (only software timer set)
1 - real time clock set

Example:
This SCLOCK command sets the real-time clock (if one is present) and the software seconds timer on the station to 5:10 AM July 31, 1991. The time is always sent in big-endian format, <MSB>...<LSB>.

Command: (13 characters)
<')'><11><255><255><0><5><24><40><150><70><45><146><247>
Acknowledge Reply: (13 characters)
<')'><8><255><255><0><5><1><4><11><201>
Clock status byte value of 4 indicates a user slave task responded to the command.

SETD Command byte value = 17

This command is used to turn on (set = 1) only specific discrete values (bits). A 1 (ON) bit in the data field will turn on the corresponding discrete value. A 0 (OFF) bit in the data field will have no effect. This command has the effect of doing a bitwise OR of the discretes specified with the data field.

Command: (14 characters + data)
<lead><length><dest><source><session><sequence><SETD><type>[start][num]...<data>...[crc]
Acknowledge Reply: (14 characters)
<lead><length><dest><source><session><sequence><ACK><type>[start][num][crc]

Example:
This SETD command sends a mask of 8 discrete outputs (type 1) starting with Y0. Bit positions Y1, Y3, Y5 and Y7 will be turned on. Other positions will be unaffected and will retain their original state. The data value is the sum of the bit positions of all outputs, in this case 170.

Command: (15 characters)
<')'><13><255><255><0><5><17><1><0><0><8><0><170><133><5>
Acknowledge Reply: (14 characters)
<')'><12><255><255><0><5><1><1><0><0><8><0><38><255>

VERS Command byte value = 3

This command sends and returns product code and software version information. This command can be used to distinguish among IOMUX, SIXTRAK, and non-SIXNET stations. It is also useful as a diagnostic tool since it can detect the presence of old software or firmware versions. All SIXNET products have a standard product code. User's Universal Protocol master or slave driver code should be assigned a code of 0 or 254 to distinguish them from standard SIXNET products. In SIXNET stations, the major id and minor id bytes (major.minor) refer to the firmware level. The assigned product codes are:

Product	Code
Non-SIXNET with old Universal Protocol capabilities (i.e. doesn't support non-byte boundaries)	0
IOMUX and Versamux	110
60-IBM/N	140
SIXTRAK Gateway or VersaTRAK	
non-programmable	200
programmable	205
RemoteTRAK and EtherTRAK I/O modules	220
SIXNET Control Room Software	250
Non-SIXNET with new Universal Protocol capabilities (i.e. supports non-byte boundaries)	254

Command: (13 characters)
<lead><length><dest><source><session><sequence><VERS>[prodcode]<major id><minor id>[crc]
Acknowledge Reply: (13 characters)
<lead><length><dest><source><session><sequence><ACK>[prodcode]<major id><minor id>[crc]

Example:
This VERS command sends product code and version information from the sending station. A product code of 254 and a version of 1.1 is sent. The responding SIXTRAK gateway replies with its product code (200) and its firmware level (1.11).

Command: (13 characters)
<')'><11><255><255><0><5><3><0><254><1><1><139><160>
Acknowledge Reply: (13 characters)
<')'><11><255><255><0><5><1><0><200><1><11><25><108>

2.4 I/O Type Numbers

It is adequate in most systems to only report four I/O types: discrete and analog inputs and outputs. By including floating point numbers and long integers (32 bit words), the vast majority of systems are covered. Use the type numbers listed below to read the default I/O types assigned in all SIXNET stations (SIXTRAK, IOMUX, Versamux, etc.). Please note that the floating point and long int registers require declaration by the user in an application program running in the station and are not present by default. A more detailed explanation of the expanded capabilities of the Universal Protocol I/O types may be found in section 3.4.

I/O Type	size	<type>	
discrete inputs	1 bit	0	Note 1
discrete outputs	1 bit	1	Note 1
analog inputs	16 bits	0	
analog outputs	16 bits	1	
long int inputs	32 bits	20	
long int outputs	32 bits	21	
floating point inputs	32 bits	22	
floating point outputs	32 bits	23	

Note: 1. Discrete I/O should be only addressed using the GETB, PUTB, GETD, PUTD, CLRD, SETD and GETS commands.

2. 32 bit counter input registers are reported as a pair of analog registers. Long int registers are not required and are not used to read these values. Read the documentation for specific SIXNET counter products for register assignments.

Section 3 _____

Advanced Topics

3.1 Computing the CRC Value

A two byte error checking code will be attached to all messages and replies. All characters are included in this check except the lead character, the CRC bytes, and the optional terminator characters. The two byte code is always sent in high byte then low byte order.

After the CRC is initialized to CRC_INIT, the crccitt() function below is used to adjust the CRC value for each data byte sent. The ones complement of the resulting two byte word is placed on the end of the message as the CRC word. The process is repeated at the receiving end, except that the incoming CRC word is added into the calculation. If the result compares to CRC_MAGIC, then no errors have occurred.

The crc value is kept in an unsigned short. When the crc contains only zero bits, zero data bytes will have no effect on it. We therefore initialize the CRC value to CRC_INIT (0xffff) so that extraneous leading zeros will be detected.

crccitt This function updates a CRC value, using the CRC-CCITT CRC polynomial, for a given data byte. The CRC value of type unsigned short should be initialized to the value CRC_INIT found below and in crc.h. The function crccitt() should then be called for each data byte included in the CRC. The sending station then sends the ones compliment of the resulting CRC value, high byte first. The receiving station includes the CRC bytes in it's CRC computation and compare the result with the value CRC_MAGIC found below and in crc.h.

CRC_INIT is FFFFh (65535 decimal)
CRC_MAGIC is 1D0Fh (7439 decimal)

```
#include <crc.h>      /* supplied with the sample program */
unsigned crccitt(
  unsigned crc,       /* old crc value */
  unsigned char data  /* data byte     */
);
```

Return Value : new CRC value

Example:

```
/* This code is for the sending station */
#include <stdlib.h>
unsigned crc;   /* crc value      */
char buf[100];  /* buffer to send */
int i;          /* work variable  */
...
crc = CRC_INIT; /* initialize crc */
for (i=0;i<100;i++) {
  crc = crccitt(crc,buf[i]);
  /* send buf[i] here */
}
```

```
crc = ~crc;     /* do ones compliment */

/* send high byte of crc, (crc >> 8)  */
/* send low byte of crc, (crc & 0xff) */

/* This code is for the receiving station */
#include <stdlib.h>
unsigned crc;   /* crc value      */
char buf[102];  /* receive buffer */
int i;          /* work variable  */
...
crc = CRC_INIT; /* initialize crc */
for (i=0;i<102;i++) {
  /* receive buf[i] here */
  crc = crccitt(crc,buf[i]);
}
if (crc == CRC_MAGIC)
  /* crc check ok */
else
  /* an error was detected */
```

A source code version of the crccitt() function is included with the Universal Protocol sample program.

3.2 Restricted Message Lengths

The length of individual Universal Driver messages is limited since the length byte cannot be more than 255. This limits the amount of data that can be contained in one message. Multiple messages should be used to transfer large amounts of data.

Message length is restricted to limit the recovery time if an error occurs. Also, the communications buffers in some SIXNET stations and in some computers have a limited size. The use of short messages allows whole messages to fit within the communications buffers and eliminates lost messages when processing delays occur.

3.3 Data Order (Format of Numeric Fields)

The most significant bit of the TYPE byte in Universal Protocol messages determines the format of multiple byte data fields. If the most significant bit is 0, little-endian format (used by Intel 80x86 processors) is used. This sends the least significant byte first, as it is stored at a lower address than the most significant byte. If the most significant bit is 1, big-endian format (used by Motorola 68xxx processors) is used. This sends the most significant byte first, as it is stored at a lower address than the least significant byte. All SIXNET stations can accept either order.

LITTLE_END_FORMAT 0

For a master I/O driver on a processor where little-endian data is most convenient (Intel 8088 - 80486, Pentium, ...), no action is required.

BIG_END_FORMAT 128

For a master I/O driver running on a processor where big-endian data is desired (Motorola 68xxx, all SIXNET stations), add 128 (0x80) to the type byte of the message.

3.4 Advanced I/O Types and Multiple Arrays

User defined I/O types and not supported in SIXTRAK and VersaTRAK stations and it is recommended that only the standard I/O types listed in section 2.4 be used for new driver development. User defined I/O types are supported by the SIXNET Control Room software for compatibility with older systems. Please note that some Windows applications that support Control Room DLL calls do not recognize user defined I/O types.

3.5 Maximum Number of I/O Registers

Each I/O type can contain up to 65535 registers. In practice, memory considerations limit this number. For example, 65535 analog registers will occupy 128K of memory.

Note: The SIXNET Control Room software, an implementation of this Universal Protocol, limits each I/O type 64K of RAM for compatibility with 16 bit systems. This, for example, limits the number of analog input registers to 32,000. We suggest that you do this also.

3.6 SIXNET Universal Protocol over Ethernet

You can communicate with SIXTRAK Ethernet Gateways over TCP/IP systems by using UDP (Universal Datagram Protocol).

The Ethernet transport layer provides highly reliable error checking. We recommend that you turn off active CRC checking in the Universal Driver messages to save processing time. (See section 2.2.1). The SIXNET Universal Protocol handles message sequencing, and acknowledgment, and will communicate reliably without the similar functions provided by the full TCP protocol.

If your driver is only intended to support EtherTRAK I/O, you may consider using the IOXCHG command. This special command update all inputs and outputs in one I/O module with a single command.

A file named `ethernet.txt' is provided on the ST-UDR disk to provide additional information.

Section 4 _____

ST-UDR Support Disk

4.1 Sample Program

A DOS sample program is supplied to help you implement the SIXNET Universal Protocol. This sample program updates and displays a block of discrete, analog and floating point inputs and outputs.

Once running, this program will continuously poll and update discrete, analog, and floating point input and output values. It will also allow the user to enter an output value for any of the three I/O types.

It is suggested that you simulate SIXNET I/O using the I/O simulator software supplied on the ST-UDR support disk. See section 4.2 below.

Refer to the file UDRDEMO.TXT for more information.

4.2 I/O Simulator

The Driver Development Kit includes an I/O simulator program to help you test your new I/O driver. This software, created and certified by SIXNET, turns any Windows based computer into a SIXNET I/O system with a large number of discrete, analog, long int, and floating point registers. This simulator responds as if it were a large network of SIXNET RTUs or SIXTRAK I/O stations.

This simulator turns any computer in your office into a permanent support tool for your driver.

You don't need SIXNET hardware to design, test or support your SIXNET driver. This simulator does it for you.

First, you must install the SIXNET Control Room on this test computer. Then, from within Windows, run setup from the simulate directory on the ST-UDR disk:

```
a:\simulate\setup
```

Note: You can use the automatic installation feature included in the SIXNET CD ROM supervisory program.

A 'Simulate' icon will be added to the SIXTRAK program group established when the Control Room software was installed. The Control Room will run for one hour in demo mode, even if you do not have a registered copy.

Connect a cable between your target system (with your new driver) and this simulator. A RS232 null modem cable is available from SIXNET (part number ST-CABLE-PF). If you are running an Ethernet based system, you may run over your existing Ethernet network.