Sixnet

Technical Note:
Writing Client Transfer Driver

Writing a Driver to Accept a Datalog Client Message

> **Note:** The SIXNET I/O Tool Kit now includes a Datalog Server driver that accepts logged data from a SIXNET client and creates ASCII text files from the data. This server software should be suitable for many applications involving client initiated data transfers. The following information is provided for software developers who wish to write their own datalog server software.

When a SIXNET station acts as a datalog client, a simple driver is needed at the server end to parse the received messages, store the data into the database in a manner appropriate for the application, and acknowledge the message. It is also possible for the reply message to synchronize the clock in the remote station and assign a new reporting time by passing parameters back in the acknowledgement reply.

Please note that programmers familiar with the SIXNET Universal Driver will find that this documentation suggests a few new simplifications in the message header. These innovations are fully compatible with all existing data formats for all existing Universal Driver messages. Please be assured that if a slave driver is implemented to receive these datalog messages, no other considerations for future compatibility with expanded capabilities need be considered. This is the only message that needs to be implemented to receive datalog records from a SIXNET client.

Datalog Client Message Definition

A SIXNET RTU, acting as a client, will transmit one or more consecutive datalog records (transactions) in a single Universal Protocol message as defined below. A receiving server only needs to receive, parse, and acknowledge this one message to accept these transactions. Since each message is self-defining, complete, and stand-alone (not dependent on any previous message), messages from any number of client stations can be received by a single server interface and processed in sequence.

In summary, the content of this datalog client transfer message is:

Message Header	Format and communications station addressing information (see below)
Message Command	Defines the message type (content)
File Identifier	It is possible to have multiple datalog files in the same RTU
Time Sent	Time-of-day of transmission &endash; especially useful to verify that the RTU clock is accurate
Record Number	Datalog records are sequentially numbered to detect missing or duplicated transmissions
Number of records	Multiple records may be sent in one message (maximum message length is 255 bytes)
Time Count	Number of bytes of data in each time field (4 is typical, can be 6 to include milliseconds)
I/O Count Fields	Number of float, long, analog (16 bit), and discrete (8 bits) variables in each record
Data Records	Timestamp followed by I/O fields as defined by the count fields above
CRC	Two byte error checking field (may be a fixed value on Ethernet systems to save time)

SIXNET Universal Protocol messages are sent in this general format:
<lead>[length][destination station number][source station number]
<session><sequence><command>...0 or more bytes of data...[crc]

Throughout this specification, '<>' brackets indicate a one byte value, '[]' brackets indicate a two byte value, and '{}' brackets indicate a four byte value. Multi-byte values are sent in big endian order (most significant byte first).

Since 'lead' defines the format and can be represented in 7 bits it is never encoded as hex characters.

Example: a Hex format NOP (no operation command, command 0) is]0009603F603F001500FA4C

] - Hex Format
0009 - length of the message beyond this field (not counting the format character and length field)
603F - for any station
603F - reply to any station
00 &endash; session 0
15 &endash; sequence / identifier number 0x15
00 - NOP command
FA4C - CRC

All implementations should process all three formats, except for Ethernet (TCP/IP) ports, which do not support Hex Format to allow higher speed, since packing/unpacking takes a lot of time.

Transmission Format (1 byte)

The lead character defines the transmission format.
Three message formats are supported:

1. Binary Format (lead character ')'):

   This is the most common format in existing applications: 8 data bits per byte.

2. Hex Format (lead character ']'):

   To allow transmission through media that either does not support 8 bit data characters or when software handshaking is required (binary data characters could be mistaken for ^S and ^Q). Data is sent as ASCII '0'-'9', 'A'-'F', or 'a'-'f' characters. The hex option 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 (e.g. 29 (decimal) is sent as "1D"). The value of the Message Length byte does NOT change to reflect the doubled number of characters sent.

3. Fixed CRC (lead character '}'):

   A binary mode that omits the CRC computation to save CPU time at both ends, this is only recommended over transport layers that already provide error checking, such as Ethernet, although it is available on serial ports. In Fixed CRC mode the [crc] field is always set to CRC_MAGIC (0x1D0F), sent <0x1d> first, <0x0F> second.

Message Length (2 bytes)
The 'length' byte defines the length of the message as an aid to processing and to potentially speed resynchronization. Its value is the number of bytes (not allowing for hex encoding) that follow the length byte, including the CRC. The minimum possible value is 9; the maximum is 257.

The physical length of a message is limited to 260 bytes including the length byte and lead character, or 519 bytes to represent the same message in hexadecimal format.

Source and Destination Addressing (2 bytes each)
Both the source and destination addresses are declared in each message so that replies can be sent over networks and complex routing schemes.

The station numbers are 16-bit values. The valid numbers are 0-16383 (station numbers) and the special value 24639 (0x603f), which is accepted by any station and may be used as the source address from drivers that have no station number.

Some Sixnet software products arbitrarily limit the station number 15999, but all implementations should support up to and including 16383.

Only messages which match the number (or numbers) assigned to a station generate replies. Do not reply to a message if the station number does not select your station.

Session Number (1 byte)
Accommodates multiple distinguishable links between two stations, this is normally 0 or 1 for this command, but other sessions may be used and the acknowledge reply ***must*** reference the same session. Only 0 through 127 and 192 through 255 are valid.

Identification/Sequence Number (1 byte)
The message identification number is referenced in replies. The identification number is echoed in the reply, so it is useful for matching replies to commands. It may be used as a sequence number, which is incremented for each successive message, but the responding station does not require the number to be unique or in any particular order.

Sub-Commands
To conserve primary command usage and help identify and group related commands, the use of subcommands has been implemented. The DLOG_NEW_RECORDS command is a DLOG (Datalog) command, DLOG_NEW_RECORDS subcommand.

27	DLOG SIXTRAK	Datalogging commands

ACK - Acknowledge valid message (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, but the data (if any) is sent after the ACK command and before the CRC.

Only generated as a reply to a message. It never generates an ACK or NAK reply.

command data: varies with command being acknowledged, see details below.

NAK - Negative Acknowledge (2)

This message acknowledges that a valid message was received (format and CRC OK, station number selects the station), but was not processed.

Possible reasons:

- unrecognized command
- unimplemented command
- could not complete the commanded action
- session number selects unknown session
- The I/O data referenced is out of range for the station.

No command returns data in a NAK.
Only generated as a reply to a message. It never generates an ACK or NAK reply.

DLOG - SIXTRAK Datalogging commands (27)

This command is used to access the SIXTRAK datalogging. It supports a number of sub-commands that allow specific datalogging actions.
command data (variable length): <sub-command>{file alias}&ldots;<data>&ldots;
reply data (variable length): <error code>...<data>...

List of Relevant Sub-commands:
DLOG_NEW_RECORDS Sub-command value: 16 Send records without request
(Unsolicited client report of datalog information)
Note: All other sub-commands are used by Sixlog Windows software to interrogate the station. They are not issued by RemoteLog or any other SIXNET remote station and need not be considered or responded to in slave drivers receiving data from a remote Client.

DLOG_NEW_RECORDS (16)

This command (added for RemoteLog) sends records from a station doing datalogging. It is unusual in that this message is sent by the station based only on data being available and the configuration calling for exporting that data. The other datalog commands are sent to request data, and the datalogging station sends the information in ACK (acknowledge) replies. The message is repeated until acknowledged by the receiving station or until the sending station has retried it enough times. Note - all multi-byte data is in big-endian (most significant byte first) format.

command data (variable size): <DLOG_NEW_RECORDS><format>[file identifier]{time sent} {record number} <number of records> <time count> <float count> <long count> <analog count> <discrete count> (&ldots;<data>&ldots;)
reply data (14 bytes): <reply format><number acknowledged>{first record}{time set} {next report}

format	Specifies format of rest of command, must be 1 (only format defined so far)
file identifier	16-bit identifier, specifies the source of the information. For RemoteLog, this is the either the station number or the value of the first configuration register.
time sent	The clock date and time (seconds since January 1, 1970) when the command was sent. This can be used to verify the time in the station so it can be corrected if necessary.
record number	The 32-bit record number of the first record being reported.
# of records	The number of records reported in this command
time count	The number of bytes of time data reported; 0, 4 or 6 bytes. Note: It is generally expected that time will be reported and that 0 will not be used.

float count for each	The number of 32-bit float values reported. There are four bytes of record data register.
long count	The number of 32-bit integer values reported. There are four bytes of record data for each register.
analog count for each	The number of 16-bit analog values reported. There are two bytes of record data register.
discrete count 8	The number of discrete points reported. There is one byte of record data for each discrete points.
data &endash; registers,	each record is reported in the order: time, float registers, long registers, analog discrete values (packed 8 to a byte, lowest number register in least significant bit) up to 232 bytes of record data.
reply format	specifies format of acknowledge reply data, must be 1 (only format defined so far)

acknowledged number of records accepted (generally equal to ’number of records').

first record	First record number acknowledged (generally equal to `record number')
time set	New time for clock in station (seconds since 1970). If 0, clock is not set. Note: It is recommended that the `time sent' parameter in the command message be compared to the server's clock and a `time set' only be sent if the remote clock is out of tolerance to avoid over use of this "system resetting" operation.
next report	specifies when to send next report of logged data. 0xffffffff means as configured, 0 to 86399 specifies the time of day (in seconds) to synchronize the report. For example, if the unit is configured to report once each hour, sending a ’next report' time of 180 would cause each report to start at 3 minutes after the hour.

Example:

Reports 2 new records, each record has 4 byte timestamp, 2 16-bit analog registers and 3 discrete values. The message is to any station, from station 1, session 0, sequence number 5.
Command (48 bytes):
<')'><0><45><96><63><0><1><0><5><27><16><1><0><1><58><241>
<114><117><0><0><12><47><2><4><0><0><2><3><58><241><100>
<96><54><159><1><146><5><58><241><114><112><54><156><1>
<151><1><229><12>
which means:
<')'> binary message format
<0><45> 45 bytes (not including format and length), fixed station address mode
<96><63> to any station (special station number 24639)
<0><1> from station 1
<0> session 0
<5> sequence 5
<27> DLOG, datalog command group
<16> DLOG_NEW_RECORDS sub-command
<1> format 1
<0><1> log ID 1 (using station number in this case)
<58><241><114><117> command sent May 3, 2001 at 3:00:05 pm
<0><0><12><47> first record is 3119
<2> 2 records
<4> 4 byte record time stamp
<0> no float registers
<0> no long (32-bit integer) registers

<2> 2 16-bit analog registers
<3> 3 bits of discrete registers
<58><241><100><96> data recorded May 3, 2001 at 2:00:00 pm
<54><159> value of first analog register is 13983
<1><146> value of second analog register is 402
<5> first and third discrete inputs were on
<58><241><114><112> data recorded May 3, 2001 at 3:00:00 pm
<54><156> value of first analog register is 13980
<1><151> value of second analog register is 407
<1> only first and discrete input was on
<229><12> CRC checksum

Acknowledge Reply (26 bytes):

<')'><0><23><0><1><96><63><0><5><1><1><2><0><0><12><47><0><0><0><0>
<255><255><255><255><49><149>

which means:

<')'> binary format
<0><23> 23 bytes (not including format and length), fixed station address mode
<0><1> to station 1
<96><63> from the ’any station' you sent to
<0> session 0
<5> reply to command with sequence number 5
<1> Acknowledge command (ACK)
<1> format 1
<2> 2 records received
<0><0><12><47> starting with record 3119
<0><0><0><0> don't change time of day
<255><255><255><255> next report at configured interval
<49><149>

CRC Computation

A two byte error checking code is 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 is the value CRC_MAGIC (1D0Fh), then no errors have occurred.

The crc value is kept as a 16 unsigned value. When the crc contains only zero bits, zero data bytes will have no effect on it.

We (and almost all other implementations) 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 16 bit 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 its 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 */
```

Overview to Message Simplifications

Experienced SIXNET driver implementers will be pleased to know that the following simplifications have been implemented for this application:

1. The message length has been extended from one to two bytes. Since the only legal values of the first byte are zero and one and since no previous message could have such a short length, the first length byte identifies the new simplified format and indicates that the other simplifications listed below are in use.
2. Since the message length is now more than one byte, it is possible for message lengths to exceed 255 characters. In practice, we have not extended the amount of data permitted in a message for many reasons. What this does though is allow for the few extra bytes associated with the new two byte fields and guarantees that if messages grow (slightly) due to these few extra bytes, that no curtailment in the amount of data sent in any Universal driver message will occur.
3. The destination and source station address fields are permanently increased to two bytes to allow for full range station addressing in a simple format.

4. All three message formats (binary, hex and Ethernet fixed CRC). It is suggested that all three formats be supported by the slave to be tolerant of communication media limitations.

Including a Control Character in a Message String

Note: This information applies to RemoteLog modules only.

It is possible to insert a control character, such as a tab, into a message string or modem initialization string. The rules are as follows:

- Alpha characters when used with the caret are not case sensitive. i.e.. ^A & ^a are equivalent.
- A double caret (^^) places a caret character into the string.
- A caret followed by three digit decimal number is the equivalent ASCII character. i.e.. ^094 is the ^ char.

Note: Numbers larger than 255 are discarded. Also, numbers that are less than three digits are assumed to be the decimal number they represent.

- A caret followed by an invalid character will be discarded. The valid characters are @ [\ ] ^ _ and alphanumeric only.