Most Commonly Used Commands on the Smart-Trak® 50 Series

ASCII and ASCII 485 Command Set

Important Customer Notice

This document is for S50 firmware version 1.xx. Sierra Instruments, Inc. reserves the right to change the command set without notification to the user. However, Sierra will also make every effort to maintain backwards compatibility with previously published command sets for the S50 line. There is no implied warranty or guarantee regarding the use of this commands set. Sierra Instruments, Inc. is not liable for any damage or personal injury, whatsoever, resulting from the use of this command set.

Protocol

The communication is based on a standard serial wire port, either RS232 or RS485. All bytes are ASCII characters. There are two command styles that can be used. The default command set is non device addressable commands. The second command set is for device addressable network. (485 multi drop networks).

The default command set start with either a '?' or a '!' for read or write respectively. The next four characters define which command is to be executed, followed by an optional data section of variable length.. Two LRC bytes follow the data section and a carriage return line feed end the string. Each command string will vary in length, but can't exceed 64 bytes. Returned string can be up to 128 bytes.

The 485 addressable command set starts with a ':' follow with a two character ASCII address. The address is in hex. Legal characters are 0-9 and A-F. Next character is '?' or '!' for read or write respectively. The next four characters define which command is to be executed, followed by an optional data section of variable length.. Two LRC bytes follow the data section and a carriage return line feed end the string. Each command string will vary in length, but can't exceed 64 bytes. Returned string can be up to 128 bytes.

The RS232 port does not depend on hardware handshaking and uses only three wires on the port: transmit, receive and ground. The default port settings are : (9600,n,8,1) 9600 baud, no parity, eight bit characters, one stop bit.

The RS485 port default settings are: (9600,n,8,1) 9600 baud, no parity, eight bit characters, one stop bit.

FLOW

Description
FLOW returns the current flow value. Write command ignores any value sent to meter.

ASCII String command

'?Flow + LRC + crlf' Read command
'!Flow + value + LRC + crlf' Write command
Returns ASCII string 'Flow + value + LRC + crlf'

Example:	?Flow29crlf	LRC=29	crlf=carriage return line feed
Returns	Flow0.0007Acrlf	LRC=7A	crlf=carriage return line feed

ASCII-485 String command

': + address + ?Flow + LRC + crlf' Read command
': + address + !Flow + value + LRC + crlf' Write command
Returns ASCII string ':01Flow + value + LRC + crlf'

Example:	:01?FlowC8crlf	LRC=C8	crlf=carriage return line feed
Returns	:01Flow0.00019crlf	LRC=19	crlf=carriage return line feed

Values are a string of digits with a decimal place: '10.00'
LRC=redundancy check bytes

SETPOINT_FLASH

Description
SETPOINT_FLASH returns the current flash memory value setpoint with read command. The write command sets the flash memeory value and makes it the active setpoint. This sets the power on set point. Warning: this command should not be used for real time control. The flash memory will wear out. This command will only work on devices configured for digital control.

ASCII String command

'?Setf + LRC + crlf' Read command
'!Setf + value + LRC + crlf' Write command
Returns ASCII string 'Setf + valueSetpoint + LRC + crlf'

ASCII-485 String command

': + address + ?Setf + LRC + crlf” Read command
': + address + !Setf + value + LRC + crlf” Write command
Returns ASCII string ': + address + Setf + valueSetpoint + LRC + crlf”

Values are a string of digits with a decimal place: '10.00'
LRC=redundancy check bytes; crlf”=carriage return line feed byte.

SETPOINT_RAM

Description
SETPOINT_RAM returns the current ram memory value setpoint with read command. The write command sets the ram memory value and makes it the active setpoint. This command can be used for real time control of the set point. This command will only work on devices configured for digital control.

ASCII String command

'?Setr + LRC + crlf” Read command
'!Setr + value + LRC + crlf” Write command
Returns ASCII string 'Setr + valueSetpoint + LRC + crlf”

ASCII-485 String command

': + address + ?Setr + LRC + crlf” Read command
': + address + !Setr + value + LRC + crlf” Write command
Returns ASCII string ': + address + Setr + valueSetpoint + LRC + crlf”

Values are a string of digits with a decimal place: '10.00'
LRC=redundancy check bytes; crlf”=carriage return line feed byte.

FULL_SCALE

Description
FULL_SCALE reads the full scale value. Read and write both will read a full scale value back. The value in the write command is ignored.

ASCII String command

'?Fscl + LRC + crlf' Read command
'!Fscl + value + LRC + crlf' Write command
Returns ASCII string 'Fscl + valueFullScale + LRC + crlf'

ASCII-485 String command

': + address + ?Fscl + LRC + crlf' Read command
': + address + !Fscl + value + LRC + crlf' Write command
Returns ASCII string ': + address + Fscl + valueFullScale + LRC + crlf'

Values are a string of digits with a decimal place: '10.00'
LRC=redundancy check bytes; crlf'=carriage return line feed byte.

GAS_NAME

Description
GAS_NAME reads the gas name.

ASCII String command

'?Gnam + LRC + crlf' Read command
Returns ASCII string 'Gnam + alphaNumericString + LRC + crlf'

ASCII-485 String command

': + address +?Gnam + LRC + crlf' Read command
Returns ASCII string ': + address + Gnam + alphaNumericString + LRC + crlf'

alphaNumericString is alpha numeric string
LRC=redundancy check bytes; crlf'=carriage return line feed byte.

UNITS

Description
UNITS reads the units of the flow.

ASCII String command

'?Unts + LRC + crlf’ Read command
Returns ASCII string 'Unts + alphaNumericString + LRC + crlf’

ASCII-485 String command

': + address + ?Unts + LRC + crlf’ Read command
Returns ASCII string ': + address + Unts + alphaNumericString + LRC + crlf’

alphaNumericString is alpha numeric string
LRC=redundancy check bytes; crlf’=carriage return line feed byte.

VERSION_NUMBER

Description
VERSION_NUMBER returns the firmware version.

ASCII String command

'?Vern + LRC + crlf’ Read command
Returns ASCII string 'Vern + alphaNumericString + LRC + crlf’

ASCII-485 String command

': + address + ?Vern + LRC + crlf’ Read command
Returns ASCII string ': + address + Vern + alphaNumericString + LRC + crlf’

alphaNumericString is alpha numeric string
LRC=redundancy check bytes; crlf’=carriage return line feed byte.

SERIAL_NUMBER

Description
SERIAL_NUMBER returns the serial number.

ASCII String command

'?Srnm + LRC + crlf’ Read command
Returns ASCII string 'Srnm + alphaNumericString + LRC + crlf’

ASCII-485 String command

': + address + ?Srnm + LRC + crlf’ Read command
Returns ASCII string ': + address + Srnm + alphaNumericString + LRC + crlf’

alphaNumericString is alpha numeric string
LRC=redundancy check bytes; crlf’=carriage return line feed byte.

SPAN

Description
SPAN reads and writes the span value.

ASCII String command

‘?Span + LRC + crlf’ Read command
'!Span + value + LRC + crlf’ Write command
Returns ASCII string 'Span + value + LRC + crlf’

ASCII-485 String command

‘: + address +?Span + LRC + crlf’ Read command
': + address + !Span + value + LRC + crlf’ Write command
Returns ASCII string ': + address + Span + value + LRC + crlf’

Values are a string of digits with a decimal place: '10.00'
LRC=redundancy check bytes; crlf’=carriage return line feed byte.

ZERO

Description
ZERO set flow offset value to zero flow reading. Warning: all flow to should be shut off for an accurate zeroing of the device.

ASCII String command

'!Zero + LRC + crlf” Write command
Returns ASCII string 'Zero + LRC + crlf”

ASCII-485 String command

': + address + !Zero + LRC + crlf” Write command
Returns ASCII string ': + address + Zero + LRC + crlf”

LRC=redundancy check bytes; crlf”=carriage return line feed byte.

RESET_ZERO

Description
RESET_ZERO sets flow offset value to zero (factory default value).

ASCII String command

'!Rezr + LRC + crlf” Write command
Returns ASCII string 'Rezr + LRC + crlf”

ASCII-485 String command

': + address + !Rezr + LRC + crlf” Write command
Returns ASCII string : + address + Rezr + LRC + crlf”

LRC=redundancy check bytes; crlf”=carriage return line feed byte.

Calculating the LRC bytes.

A procedure for generating an LRC is:
Add all bytes in the message, excluding the starting ‘colon’, if using 485 addressing mode, and crlf, into a 8 bit value. All carries are discarded.

LCR = -(8 bit value) ; two's complement
Example:
Packet[10] is a 10 byte char array assigned as follows Packet[0] = '?' = 0x3F
Packet[1] ='F' = 0x46 Packet[2]='l' =0x6C Packet[3]='o' =0x6F Packet[4]='w' =0x77
Packet[5]=0 // will be high byte of LRC Packet[6]=0 // will be low byte of LRC
Packet[7]=0 // will be carriage return; 0x0D Packet[8]=0 // will be line feed; 0x0A
Packet[9]=0 // string terminator in C optional byte

packetLength = strlen(Packet) // packetLength= 5; C parses to first 0 in the array
unsigned byte LRC=0 // start with 0 LRC value
For(i=0; i<packetLength; i++) LRC = LRC + Packet[i]; // add all bytes into LRC
For this example: LRC = 0x3F + 0x46 + 0x6C + 0x6F + 0x77 = 0x1D7 (if carries are not discarded)

LRC = 0xD7 discarding the carry LRC = -LRC // two's complement
Two's complement is ~(0xD7) + 1 = 0x28 + 1 = 0x29 (The ~ is the not operator) (see Wikipedia for more on not operator or two's complement)
Now split the result into high and low bits, 2 and 9.
Convert each of these values to their ASCII hex equivalent.
2 -> 0x32
9 -> 0x39

Packet[5] = high 4 bits of LRC converted to ASCII, 2 -> 0x32
Packet[6] = low 4 bits of LRC converted to ASCII, 9 -> 0x39
Packet[7] = 0x0D // carriage return
Packet[8]= 0x0A // line feed

Recap:
1) Add all bytes in message
2) Apply NOT to result and add 1 (2's complement)
3) Split byte into high and low segments
4) Convert high and low segments to ASCII Hex value

Calculating the LRC bytes using C#.

```
private static byte[] CalcLRC(byte[] cmnd)
    {

        int i, j, bSize = cmnd.Length;
        uint lrc,high,low,CR,LF;
        string temp;
        byte[] bytes = new byte[((bSize)+4)];
        lrc=0x00;
        CR = 0x0d;
        LF = 0x0a;

        //loop through input byte array
        for (i=0; i<cmnd.Length; i++)
        {
           //sum message bytes
           lrc = lrc + (uint) cmnd[i];
```

```
        //insert each message byte into result
        bytes[i] = cmnd[i];
    }

    //remove remainder
    if (lrc > 256)
    {   lrc = lrc - (uint) (256 * Math.Floor((double) lrc/256));
    }

    //perform 2's complement on sum
    lrc = (ushort)(~lrc);
    lrc = lrc + 1;

    //remove remainder
    if (lrc > 256)
    {   lrc = lrc - (uint)(256 * Math.Floor((double)lrc / 256));
    }

    //split into high and low bits
    temp = lrc.ToString("X");
    high = (uint)temp[0];
    low = (uint)temp[1];

    //insert high, low, carriage return, and line feed into result
    bytes[(bSize)]=(byte) high;
    bytes[(bSize+1)]=(byte) low;
    bytes[(bSize+2)]=(byte) CR;
    bytes[(bSize + 3)] = (byte) LF;

    //return result byte array with input, LRC, CR, LF
    return bytes;
}
```