Most Commonly Used Commands on the SmartTrak® Series

IMPORTANT CUSTOMER NOTICE

This document is for firmware version 2.044. 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 SmartTrak®. 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 SmartTrak communication is based on a standard RS232 port. All bytes are ASCII except the two CRC bytes used for a redundancy check. Each command starts with four bytes of command type, followed by a variable length value section. The two CRC bytes follow the value string and a carriage return ends the string. Each command string will vary in length but, can not exceed 26 bytes.
Command String = {4 command bytes + variable length value + 2 CRC bytes + carriage return} < 26 bytes.

Note: a CRC of hex ffff will bypass the CRC check and will return the correct command and CRC. Do not confuse this with ff ff in ASCII, which is actually 66 66 66 66 in hex.

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

DESCRIPTION

Commands for the Meter. There are SmartTrak commands (version 1.xx), SmartTrak 2 commands (version 2.xx), and SmartTrak 100 commands (2.xx). The 2.xx commands will support some of the 1.xx commands. If a 1.xx commands is not shown below, it is not supported in the 2.xx commands. CMNDS is a class included in the class of Meter. This page shows all types, fields, and values as derived from Meter. Version 2.xx commands that start with a '?' are read commands and commands that start with a '!' are write commands. This is list of the most commonly used commands, which are relative safe.

FLOW

DESCRIPTION
FLOW makes the streaming mode data = flow only. If meter is in streaming mode, this data will stream until a FLOWwSETPOINT, FLOWwSETPOINTwDAC or TOTALwDIFFwDAC is sent to the meter. If not in streaming mode then one flow will be return with each FLOW command sent. ReadMeter, WriteMeter both will set flow as the streaming data.

ASCII STRING COMMANDS

'?Flow + CRC + cr' Version 2.xx read command
'!Flow + value + CRC + cr' Version 2.xx write command
'?Flow + CRC + cr' Version 1.xx read command

Returns ASCII string 'Flow + value + CRC + cr'

Values are a string of digits and an optional decimal place: '10.00'

SETPOINT

DESCRIPTION
SETPOINT sets the control setpoint. The set point is store in the flash memory. This command is not recommended for version 2.xx,. Use SETPOINT_FLASH.

ASCII STRING COMMANDS

'?Sinv + CRC + cr' Version 2.xx read command
'!Sinv + value + CRC + cr' Version 2.xx write command
'?Sinv + CRC + cr' Version 1.xx read command
'Sinv + value + CRC + cr' Version 1.xx write command

Returns ASCII string 'Sinv + valueSetpoint + CRC + cr'

Values are a string of digits and an optional decimal place: '10.00'
CRC=redundancy check bytes; cr=carrage return byte.

SETPOINT_FLASH

DESCRIPTION
SETPOINT_FLASH returns the current flash memory value setpoint with read command. The write command sets the flash memory value and makes it the active setpoint. This would be the setpoint after a power down.

ASCII String commands

'?Setf + CRC + cr' Version 2.xx read command
'!Setf + value + CRC + cr' Version 2.xx write command

Returns ASCII string 'Setf + valueSetpoint + CRC + cr'

Values are a string of digits and an optional decimal place: '10.00'
CRC=redundancy check bytes; cr=carrage return 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.

ASCII STRING COMMANDS

'?Setr + CRC + cr' Version 2.xx read command
'!Setr + value + CRC + cr' Version 2.xx write command

Read returns ASCII string 'Setr + valueSetpoint + CRC + cr'

Write returns ASCII string 'Sinv + valueSetpoint + CRC + cr'

Values are a string of digits and an optional decimal place: '10.00'
CRC=redundancy check bytes; cr=carrage return byte.

UNIT_INDEX

DESCRIPTION

UNIT_INDEX selects the units the meter displays. The index value is between 1 and 30 and is assigned as shown. scc/s = 1, scc/m = 2, scc/H = 3, Ncc/s = 4, Ncc/m = 5, Ncc/H = 6, SCF/s = 7, SCF/m = 8, SCF/H = 9, NM3/s = 10, NM3/m = 11, NM3/H = 12, SM3/s = 13, SM3/m = 14, SM3/H = 15, sl/s = 16, sl/m = 17, sl/H = 18, NL/s = 19, NL/m = 20, NL/H = 21, g/s = 22, g/m = 23, g/H = 24, kg/s = 25, kg/m = 26, kg/H = 27, lb/s = 28, lb/m = 29, lb/H = 30

ASCII STRING COMMANDS

'?Unti + CRC + cr' Version 2.xx read command
'!Unti + unitIndex + CRC + cr' Version 2.xx write command
'?Unti + CRC + cr' Version 1.xx read command
'Unti + unitIndex + CRC + cr' Version 1.xx write command

Returns ASCII string 'Unti + unitIndex + CRC + cr'

unitIndex is an integer string value of 1 to 30
CRC=redundancy check bytes; cr=carrage return byte.

VALVE_INDEX

DESCRIPTION

VALVE_INDEX selects the state of the valve. Automatic = 1, Closed = 2, Purge = 3. This is a write to flash memory command. For version 2.xx, this command is not recommended. See VALVE_FLASH_INDEX and VALVE_RAM_INDEX.

ASCII STRING COMMANDS

'?Vlvi + CRC + cr' Version 2.xx read command
'!Vlvi + valveIndex + CRC + cr' Version 2.xx write command
'?Vlvi + CRC + cr' Version 1.xx read command
'Vlvi + valveIndex + CRC + cr' Version 1.xx write command

Returns ASCII string 'Vlvi + valveIndex + CRC + cr'

valveIndex is an integer string value of 1 to 3
CRC=redundancy check bytes; cr=carrage return byte.

GAS_INDEX

DESCRIPTION

GAS_INDEX selects the current gas.

ASCII STRING COMMANDS

'?Gasi + CRC + cr' Version 2.xx read command
'!Gasi + gasIndex + CRC + cr' Version 2.xx write command
'?Gasi + CRC + cr' Version 1.xx read command
'Gasi + gasIndex + CRC + cr' Version 1.xx write command

Returns ASCII string 'Gasi + gasIndex + CRC + cr'

gasIndex value is 1 through 10
CRC=redundancy check bytes; cr=carrage return byte.

STREAM

DESCRIPTION
Communications can be in one of three mode set by this STREAM command. In off mode, the meter will respond when queried with a read command. In echo mode, the meter will respond with either a read or a write command. In stream mode, the meter will continuously send data back. If a read or write command needs to update more than one value, all values with be sent back on write command.

ASCII STRING COMMANDS

'?Strm + CRC + cr' Version 2.xx read command
'!Strm + streamString + CRC + cr' Version 2.xx write command
Returns ASCII string 'Strm + streamString + CRC + cr'
streamString = "On", "Off", "Echo"
CRC=redundancy check bytes; cr=carrage return byte.

VERSION_NUMBER

DESCRIPTION
VERSION_NUMBER returns the firmware version. This is useful when Ver. 1.xxx and 2.xxx are in the same system to determine which command to send.

ASCII STRING COMMANDS

'?Vern + CRC + cr' Version 2.xx read command

Returns ASCII string 'Vern + alphaNumericString + CRC + cr'

alphaNumericString is alpha numeric string
CRC=redundancy check bytes; cr=carrage return byte.

SERIAL_NUMBER

DESCRIPTION
SERIAL_NUMBER returns the serial number

ASCII STRING COMMANDS

'?Srnm + CRC + cr' Version 2.xx read command

Returns ASCII string 'Srnm + SERIAL_NUMBER + CRC + cr'

alphaNumericString is alpha numeric string
CRC=redundancy check bytes; cr=carrage return byte.

SYNC

DESCRIPTION
Sync returns the following events, VersionNumberEvent, SerialNumberEvent, DataEvent, ManufactureNumberEvent, TypeEvent, PassWordEvent, SetpointEvent, FullScaleEvent, GasNameEvent(10 times for all gas names), GasIndexEvent, GasSpanEvent(current gas only), SetpointIndexEvent, OuputIndexEvent, UnitIndexEvent, ValveFlashEvent, StreamEvent, SyncEvent

ASCII STRING COMMANDS

'?Sync + CRC + cr' Version 2.xx read command

Returns ASCII string 'Sync + "" + CRC + cr'

SyncEvent returned last to indicate all events have been sent
CRC=redundancy check bytes; cr=carrage return byte.

ZERO
DESCRIPTION
ZERO sets the flow offset value to a zero flow reading. Warning: All flow must be shut off, the unit needs to be at pressure with the gas being used.

ASCII STRING COMMANDS

'!Zero + "" + CRC + cr' Version 2.xx write command

CRC=redundancy check bytes; cr=carrage return byte.

RESET_ZERO

DESCRIPTION
Reset zero flow offset value to zero

ASCII STRING COMMANDS

'!Rezr + "" + CRC + cr' Version 2.xx write command

Returns ASCII string 'Rezr + "" + CRC + cr'

CRC=redundancy check bytes; cr=carrage return byte

CRC CALCULATIONS

Below is the routine used to calculate the CRC bytes in C#.
Comments are between the /* */.

```
                private static uint CalcCRC(byte[] cmnd)
/* cmnd is a byte array containing the command ASCII string … cmnd[]=”Sinv2.000” */
/* An unsigned 32 bit integer is return to the calling program */
/* only the lower 16 bits contain the crc */
                {
                        int i,j;  /* interating indexes for the for loops */
                        uint crc;       /* crc variable that will be returned */

                        crc=0xffff;     /* initialize crc to hex value 0xffff */

                        for (i=0; i<cmnd.Length; i++)
/* this for loop starts with ASCCII ‘S’ and loops through to the last ASCII ‘0’ */
                        {
                                crc=crc^((uint)(cmnd[i]*0x0100));
/* the ASCII value is times by 0x0100 first then XORED to the current crc value */
                                for(j=0; j<8; j++)
/* the crc is hashed 8 times with this for loop */
/* if the 15th bit is set (tested by ANDING with hex 0x8000 and testing for 0x8000 result)
then crc is shifted left one bit (same as times 2) XORED with hex 0x1021 and ANDED
to hex 0xffff to limit the crc to lower 16 bits. If the 15th bit is not set then the crc is shifted
left one bit and ANDED with hex 0xffff to limit the crc to lower 16 bits. */
                                {
                                        if((crc&0x8000)==0x8000)
                                                crc=((crc<<1)^0x1021)&0xffff;
                                        else
                                                crc=(crc<<1)&0xffff;
/* end of j loop */             }
/* end of i loop */     }
/* There are some crc values that are not allowed, 0x00 and 0x0d */
/* These are byte values so the high byte and the low byte of the crc must be checked and
incremented if the bytes are either 0x00 0r 0x0d. */
                        if((crc&0xff00)==0x0d00) crc +=0x0100;
                        if((crc&0x00ff)==0x000d) crc +=0x0001;
                        if((crc&0xff00)==0x0000) crc +=0x0100;
                        if((crc&0x00ff)==0x0000) crc +=0x0001;
                        return crc;
                }
/*  If the string Sinv2.000 is sent through this routine the crc = 0x8f55 */
/* The complete command “Sinv2.000” will look like this in hex
 0x53 0x69 0x6E 0x76 0x32 0x2e 0x30 0x30 0x30 0x8f 0x55 0x0d */
```

Below is the c# routine with no comments.

```
private static uint CalcCRC(byte[] cmnd)
{
        int i,j;
        uint crc;

        crc=0xffff;
        for (i=0; i<cmnd.Length; i++)
        {
                crc=crc^((uint)(cmnd[i]*0x0100));
                for(j=0; j<8; j++)
                {
                        if((crc&0x8000)==0x8000)
                                crc=((crc<<1)^0x1021)&0xffff;
                        else
                                crc=(crc<<1)&0xffff;
                }
        }
        if((crc&0xff00)==0x0d00) crc +=0x0100;
        if((crc&0x00ff)==0x000d) crc +=0x0001;
        if((crc&0xff00)==0x0000) crc +=0x0100;
        if((crc&0x00ff)==0x0000) crc +=0x0001;
        return crc;
```

CRC ROUTINE IN MS VISUAL BASIC:

```
Public Sub SendPacket(packet As String)
Dim crcBytes() As Byte
Dim crcLen, vpacket
Dim crcWord As Long
Dim i, j

   packet = Replace(packet, ",", ".") ' if commas are used as decimal place holder
   crcLen = Len(packet) - 1    ' zero based counter
   ReDim crcBytes(0 To crcLen)
   For i = 0 To crcLen ' convert string to bytes
      crcBytes(i) = Asc(Mid(packet, i + 1, 1)) ' get character
   Next
' calc crc
      crcWord = (&H1FFFE) / 2 ' not -1
      For j = 0 To crcLen
         crcWord = crcWord Xor CLng(crcBytes(j)) * &H100
```

```
                    For i = 1 To 8
                If (crcWord And &H8000) Then
                   crcWord = (crcWord * 2) Xor &H1021 ' prime number
                Else
                   crcWord = crcWord * 2
                End If
                crcWord = (((crcWord * 2) And &H1FFFE)) / 2
             Next i
          Next j
          If (crcWord And &HFF00) = &HD00 Then    ' carriage char not allowed
             crcWord = crcWord + &H100          ' make this byte &h0exx
          End If
          If (crcWord And &HFF00) = &H0 Then      ' zero char is death
             crcWord = crcWord + &H100          ' make this byte &h01xx
          End If
          If (crcWord And &HFF) = &HD Then        ' carriage char not allowed
             crcWord = crcWord + 1              ' make this byte &hxx0e
          End If
          If (crcWord And &HFF) = &H0 Then       ' zero char is death
             crcWord = crcWord + 1              ' make this byte &hxx01
          End If

ReDim Preserve crcBytes(0 To crcLen + 3) ' make room for crc and carriage return
          crcBytes(crcLen + 1) = Int(crcWord / &H100) ' high byte
          crcBytes(crcLen + 2) = (crcWord And &HFF)   ' low byte
          crcBytes(crcLen + 3) = 13   ' carriage return
          vpacket = crcBytes
       commQueue.Add vpacket
          End Sub
```