

Format of SPRAY SBD messages

updated Dec 2006 with v.0612 changes (Engineering time-series option)

The SBD (Short-Burst-Data) message for Spray is based on the Aug 2003 data message format to the SIO ground station (GS), whose first character is an 'X'. References to the 'X message' or 'X data message' are the same as the data message. For continuity, the SBD will also maintain the 'X' delimiter for the first character, as well as the checksum at the end, allowing in principal for the same message to be sent by either SBD or GS mode.

The data is assumed to be binary and each byte can have any value from 0x00 to 0xff. The format of the message is:

Xnnmddp<data>\$cc>

- X** = the character **X**.
- nn = number of data characters in the message following after nn. The count does not include **X**, nn, or anything from **\$** to the end **>**. The count is in 2 binary bytes with MSB first and LSB second.
- mm = serial number SN of SPRAY. The SN is in 2 binary bytes with MSB first and LSB second.
- dd = the dive number in 2 binary bytes with MSB first and LSB second.
- p = one-byte packet ID index, range 0 to 255. This is used to identify multiple X messages within a dive cycle. The data for each dive cycle starts with p=0 and the data will normally fit in a single packet so typically p = 0.
- <data> = binary data characters. The length of <data> = nn -5. The contents of the <data> section is described below.
- \$** = a dollar sign delimiter at the start of the checksum
- cc = the 8 bit byte-wise checksum from **X** to the byte preceding the **\$**. The 8 bit sum is coded as 2 4bit nibbles. The binary value of a nibble is converted to a visible character by adding 0x30. Thus a value of 0x0 -> 0x30 = character '0', 0x1 -> 0x31 = '1', 0xe -> 0x3e = '>', and 0xf -> 0x3f = '?'.
> = a > delimiter at end of checksum which also serves as a prompt to the GS that the ISU is done transmitting and that the GS may now transmit to the ISU.

SAMPLE CHECKSUM C CODE can be found at the end of this document.

The remainder of this document describes the format of the <data> portion of the message sent from SPRAY to the ground station (**GS**). The format of commands from **GS** sent to Spray will be described in another document.

NOTE: All 2-byte integers sent in the SBD message are defined as the MSB first, LSB second.

The <data> section contains information from multiple sensors. Data from successive sensors are separated by a semicolon (';' = 0x 3b); the final sensor is terminated by a '\$' (immediately preceding the \$ delimiter).

IDjj<sensor_data>;
ID = one-byte sensor ID code.
jj = Number of bytes for this sensor. The count includes **ID**, **jj**, and the trailing ;.
 The count is in 2 binary bytes with MSB first and LSB second.
 <data>= data bytes. The length of <data> = jj bytes, and its contents are described below for each sensor type.
 ; = delimiter at the end of each sensor's data.

The **ID** byte is divided into two 4-bit nibbles. The MS nibble identifies the sensor and the second nibble allows for specifying alternative coding schemes for that sensor:

Sensor	ID byte(hex)	
GPS	00	fix at start of mission
GPS	01	fix at start of dive
GPS	02	fix at end of dive
GPS	03	fix following mission abort
Pressure	10	time series of pressure (scaled 1st difference)
Temperature	20	time series of temperature (scaled 1st difference)
Conductivity	30	time series of conductivity (scaled 1st difference)
Optical	40	time series of optical data (scaled 1st difference)
Eng. Time-Series	cx	(starting code v.0612)
Route	d1	('d1' is updated version, starting code v.0608)
Waypoint List	d2	(starting code v.0608)
Dump of the EEPROM settings	d3	(starting code v.0608)
Echo of Received Shore Cmds	de	(starting code v.0608)
Engineering	e0	engineering test & diagnostic data
Test pattern	fn	test pattern, n=0..f used as test id

FEB 2006 ADP Update

see ADP_SBD_FORMAT.DOC for more details

Acoustic Doppler Profile (ADP)	5x	all ADP time series sent as scaled 1st difference.
ADP Pressure	50	depth series of each ensemble average.
ADP East Velocity Profile	51	east component profile (mm/s, >0 for east flow)
ADP North Velocity Profile	52	north component profile (mm/s, >0 for north flow)
ADP Vertical Velocity Profile	53	vertical velocity component (mm/s, >0 for upwards flow)
ADP Amplitude	54	last depth bin's amplitude in each ensemble, averaged across beams, 1 count = 0.43 dB.
ADP pitch	55	pitch at each ensemble, 1 count = 0.4 degrees.
ADP roll	56	roll at each ensemble, 1 count = 0.4 degrees.
ADP heading	57	heading, 1 count = 0.1 degrees.
ADP Backscatter Profile=ADP_B	58	depth series of beam-averaged backscatter.
ADP Parameters	5F	The present ADP settings (NEW JAN06)

On ascent, for every CELL_SIZE meters (nominally 4-5 m), a 16-ping ensemble ADP average is performed, with pressure, pitch, roll, and heading recorded. At the end of ascent the ADP first-differenced velocities are averaged over adjoining depths and then re-integrated before data packing. All of the ADP time series use the same scaled first-difference packing algorithm as used for Pressure, Temperature, and Conductivity.

The depth of the first velocity estimate is given by: $z = \text{ADP_P}[0] + \text{BD} + 2 * \text{CELL_SIZE}$
 where BD = blanking distance (nominally 2 m), and CELL_SIZE is the ADP cell size (range bin size). The factor of 2*CELL_SIZE occurs due to integration of shear to give velocity always initializes the first cell's velocity to zero, and then integrates from there. Since the first cell is always zero, it is not transmitted, and the profile starts with the second cell's value.

GPS data (ID=0x00, 0x01, 0x02, 0x03)

The LS nibble of the ID indicates in what phase of the mission the fix was taken. The remainder of the data is the same for all mission phases. The length of GPS data is in bytes 1 and 2. GPS fix data starts in byte 3:

Byte	Contents
0	Mission phase: 0 = start of mission 1 = start of this dive 2 = end of this dive 3 = following mission abort
1-2	Number of bytes in the data, 23 = 0x17 with the format as described here
3	0 if fix is invalid, +1 if longitude is East, -1 if longitude is West
4	Signed latitude degrees, >0 = North, <0 = South, range +-90 degrees
5	Integer latitude minutes (unsigned), range 0 to 59
6	Fractional latitude minutes (unsigned) in .01 degrees, range 0 to 99
7	Unsigned longitude degrees, range 0 to 179 degrees
8	Integer longitude minutes (unsigned), range 0 to 59
9	fractional longitude minutes (unsigned) in .01 degrees, range 0 to 99
10	Wing index and Roll Status. See comment below.
11	MS byte of GPS week (GPS week =0 to 1023 in 10 bits)
12	LS byte of GPS week
13	GPS day of week, 0=Sunday, 6=Saturday
14	UTC hour
15	UTC minutes
16	Time to get fix = (seconds to get fix)/10, range 0 to 255 = 0 to 2550 seconds
17	MSnibble=status, LS nibble = number of satellites in view Status 0x1 = almanac is NOT complete Status 0x2 = no battery backup for RAM Status 0x4 = health =0x01 thru 0x09 from GPS receiver Status 0x8 = antenna bit fault
18	Minimum signal level
19	Average signal level
20	Minimum signal level
21	10*horizontal dilution of precision
22;	terminator

Wing and Roll Status Byte (byte 10) Updated Jun06

- b0 = is set to the GPS wing (1=port,0=stbd) for the GPS message in lifo_gps()
- b1 = 1 = roll function timed out (slow motor/bad pot)
- b2 = 1 = failure in the last 2 s (slow motor/bad pot).
- b3 = 1 = bad potentiometer (pot is out-of-bounds by over 200 counts).
- b4 = 1 = starboard wing is not at the correct position.
- b5 = 1 = port wing " ".
- b6 = 1 = zero wing (wings flat) " ".
- b7 = 1 = intermittent pot behavior (the pot initially reads in-bounds, and then bad).

Pressure data (ID=0x10)**Temperature data (ID=0x20)****Conductivity data (ID=0x30)****Optical data (ID=0x40)**

Profile data from the pressure, temperature, conductivity, and optical sensors are all processed in the same way and the message format differs only in the ID code. The pressure data is examined to find where the SPRAY leaves the surface on its descent and any surface data is ignored. The time series of data from all channels is processed in the same way. First the data is block averaged in such a way as to result in about 200 block averages of each channel. The block averages of each channel are synchronous with each other. Then the time series is broken into sub-blocks of 20 samples, and a first-differencing method is applied to each sub-block to reduce the number of bytes required to transmit the data. The final sub-block of the time series may have fewer than 20 samples in it. The data message looks like:

IDjj<sub-block 0><sub-block 1> . . . <sub-block m>;
ID = one-byte sensor ID code.
 jj = Number of bytes for this sensor. The count includes **ID**, jj, and the trailing ;. The count is in 2 binary bytes with MSB first and LSB second.
 <sub-block i> = first-differenced data from the ith sub=block where i=1,...,m =number of sub-blocks. If i<m, the sub-block will have 20 values in it and will have a total length of 22 bytes. The mth sub-block will have between 1 and 20 values and a length between 3 and 22 bytes.

Suppose a sub-block has the n values $v[0], v[1], \dots, v[n-1]$. Then this sub-block will be transmitted as:

Sub-block Byte	Contents
0	one-byte scaling factor S, range = 1 to 255. S is chosen so that the scaled first-differences fit in one byte.
1	MS byte of $v[0]$
2	LS byte of $v[0]$
3	LS byte of $\{ v[1] - v[0] \}/S$
4	LS byte of $\{ v[2] - v[1] \}/S$
...	
n+1	LS byte of $\{ v[n-1] - v[n] \}/S$

Engineering Time-Series data (ID=0xCn) (updated starting v.0612)

The engineering time-series data are processed using the same algorithm as the profile data (see the previous page). The ID byte is comprised of '0xCn' where n is the sensor id:

- 0 time [s] since start-of-dive (wing down and valve opened).
- 1 Pressure [counts] (process the same as the profile data).
- 2 Heading [degrees*10] from the compass.
- 3 Pitch Angle [degrees*10] from the compass tilt sensor.
- 4 Roll Angle [degrees*10] from the compass tilt sensor.
- 5 Pitch potentiometer [counts].
- 6 Roll potentiometer [counts].

The time-series is a decimated sub-sample of the 8-s raw scans, where the decimation = ndec, and the selection of the desired returned sensors, are controlled by the shore command 'H' (refer to that documentation for further details). These time-series allow further diagnosis of:

- a) Descent speed.
- b) Heading control.
- c) Health of pitch and roll sensors.
- d) Health of pitch and roll mechanisms.

Route data (ID=0xd1) contains Route settings and steering-mode values
(updated starting v.0608)

Byte	Contents	
0	ID	= 0xd1 = identifier that route data follows.
1	N_msb	= # of bytes for the route data (msb)
2	N_lsb	= # of bytes for the route data (lsb). Total = N_msb*256+N_lsb.
3	N_route	= Total # of entries in the route
4	Now	= present route entry that we are now heading for.
5	End_Action	= Action to take at the end of the route: n=0 : go to the HOME waypoint. n=1 : repeat route from the beginning. n=2 : reverse the route (trace the route back to waypt 1). n=3 : stay stationed at the last waypoint. n=4 : ABORT when the last waypoint is reached.
6	Dir	= direction to step through the route 1 = forward (go to waypoint 1->2->3...) -1 = Reverse (go to 3->2->1)
7	Use_Buck	=1 if using current-bucking mode, =0 if NOT. If ON, then the Spray heading is adjusted for set correction. This mode has no timeout (it is either on or off).
8-9	Use_Cross	=0 if current-crossing mode is off, else Cross = relative angle to the current to steer (-90 = to the left 90 degrees, +90 = to the right).
10-11	Cross_TMO	=Dive Number when Use_Cross will no longer be used.
11-12	Man_Theta	=-1 = Off, -2 = just do circles, >=0 = manual heading [TRUE] to steer.
13-14	Theta_TMO	=Dive Number when Man_Theta is no longer used.
15-16	Steer_Pt	=Steering Point Distance [km] further up the route to head for. This advanced feature is to get the Spray back on track faster.
17-18	Steer_TMO	=Dive Number when Steer_Pt is no longer used.
19-20	CMin	For (Buck, Cross, Steer_Pt) this is the minimum correction allowed to the heading. New > (uncorrected + Cmin).
21-22	Cmax	The maximum allowed course correction for (Buck, Cross, Steer_Pt).
byte	This is followed by the info for each route entry	
22 + (j*5)	j = route entry #	
+1	i_waypt	route[j] points to waypoint index i_waypt in the waypoint list.
+2	detect	=Arrival-Detect mode (0=range only, 1=range+finish line, 2=finish only).
+3	Range	=Watch-Circle radius for range-detect [km] (0=auto-range).
+4-5	Approach	=Approach Angle [TRUE] to the next route entry: used to define the finish line angle. If =0, then it auto-computes.

Waypt data (ID=0xd2) contains the Waypoint List (updated starting v.0608)

Byte	Contents	
0	ID	= 0xd2 = identifier that waypoint data follows.
1	N_msb	= # of bytes for the waypoint data (msb)
2	N_lsb	= # of bytes for the waypoint data (lsb). Total = N_msb*256+N_lsb.
3	N_Waypt	= Total # of waypoints in the waypoint list
byte	This is followed by the info for each waypoint entry.	
3 + (j*8)	j = waypoint # j=0 = HOME waypoint	
+1	Lat_deg_msb	
+2	Lat_deg_lsb	Lat_deg = Lat_deg_msb*256 + Lat_deg_lsb
+3	Lat_dx_msb	
+4	Lat_dx_lsb	Lat_dx = Lat_dx_msb*256 + Lat_dx_lsb
+5	Lon_deg_msb	
+6	Lon_deg_lsb	Lon_deg = Lon_deg_msb*256 + Lon_deg_lsb
+7	Lon_dx_msb	
+8	Lon_dx_lsb	Lon_dx = Lon_dx_msb*256 + Lon_dx_lsb

The above is repeated for each waypoint in the list.

To compute decimal degrees latitude/longitude, apply the following algorithm

Lsign = 1 if Lat_deg>0, else -1. (Lat_deg is a signed value of degrees)

xlat = Lsign*(abs(Lat_deg) + Lat_dx*0.001). (Lat_dx is an unsigned fractional value)

EEPROM data (ID=0xd3) Is a list of all the Shore-resettable parameters (**updated starting v.0608**)

Byte	Contents
0	ID = 0xd3 = identifier that eeprom data follows.
1	N_msb = # of bytes for the eeprom data (msb)
2	N_lsb = # of bytes for the eeprom data (lsb). Total = N_msb*256+N_lsb.
3	EE_VER = EEPROM version (608, 610,) list follows, containing 1 byte addr, 2 bytes value
3 + (j*3)	j = index entry
+1	addr =address : used to link the address to the label in the .cfg file. the addr-name link is also given by the eeprom dump when directly communicating to the Spray (E.eeprom, D.isplay all).
+2-3	value

ECHO data (ID=0xde) Is the echo of any received shore commands (**updated starting v.0608**)

Byte	Contents
0	ID = 0xde = identifier that echo data follows.
1	N_msb = # of bytes for the echo data (msb)
2	N_lsb = # of bytes for the echo data (lsb). Total = N_msb*256+N_lsb.
3->end	character string of all commands received during the present surface communications.

When the Spray has completed sending all of its dive messages, and has checked for all shore commands, if it has received any, it will send one last message that is just the echo of all of the received commands.

Engineering data (ID=0xe5)**0xe5 is used for v0610**

NOTE: This structure is defined in terms of the Motorola integer format of BYTE 1=MSB, BYTE 2=LSB. If the data is being processed on a PC (integer = BYTE 1=LSB, BYTE 2 = MSB) then a work-around is required.

The engineering data is used in diagnosing glider behavior and anomalies. The contents of the engineering message is described in the SPRAY code by a struct eng_param. The easiest way to parse and understand what the engineering data bytes mean is to copy the struct from the SPRAY code and copy it into the processing code. Then the processing code can refer to each parameter by name to parse the message.

The declaration of the struct includes the sensor ID=0xe0 in parameter eng_param.isu_type and the ';' terminator in the character eng_param.end. Two dummy parameters have been added to the struct to maintain its length as an integer number of words and to insure that eng_param.isu_type is contiguous to the MS byte of eng_param.nbytes. The portion of the struct from eng_param.isu_type through eng_param.end is copied into the buffer for the X message.

Version 0610:

```

struct eng_param { // parameters for engineering data
  char dum1;      // dummy byte to get isu_type on even boundary.
  char isu_type;  // first char = isu identifier = 0xe5
  short nbytes;  // #bytes in the eng_param structure
  short Zmax;    // max depth of dive, dBar
  short alt;     // altimeter counts at turning depth (see below)
  short bat;     // avg battery [V*100] in bottom turn
  short current; // avg current {amps*100} in bottom turn
  short Psurf;   // avg surface press at the start of the dive
  short pitch;   // pitch, degrees, of dive
  short head;    // heading steered; [TRUE] as of v0608, 0610
  short drx;     // dead-reckoning in the x direction
  short dry;     // dead-reckoning in the y direction
  short ydeg;    // way-point degrees latitude
  short dy;      // way-point fractional latitude x 1000
  short xdeg;    // way-point degrees longitude
  short dx;      // way-point fractional longitude x 1000
  char n_badamp; // #times had too high of current,
  char navg;     // #points averaged for profile (changed to char jun06)
  short ti_pump; // pump time, msbyte = pump deep going down, lsb=total ti.
  short vac;     // vacuum ~ = ( in-Hg)*100
  short idive;   // present dive/profile number
  short miss_id; // mission id
  short max_amp; // time/10 (msb) + max amps (.02 res, lsb) during bottom turn
  char r_err;    // jun06 : accumulated roll error (was old n_badamp byte)
  char t_SBD;    // time trying to get SBD message through, [s]/10
  char ntries;   // #connect tries from the LAST dive
  char nsent;    // #packets sent from the LAST DIVE
  char sbdi_stat; // SBDI= out-bound transmission status: wing in upper nibble
  char sbd_shore_stat; // status of parsing a shore-command
  short exc_stat; // New v0608, exception status word
  short surf_tm;  // New v0608, LSB =time[s/10] to leave surface to 2m depth.
                  // MSB =time[s/10] at surface end-dive until GPS is on.
  char end;      // end character for isu packet
  char dum2;     // end on even-boundary
}; // end eng_param

```

Engineering Definitions

Zmax [dBar] The maximum depth observed in the bottom turn (transition from descent to ascent).

Alt altimeter parameter. For ISU_Type = 0xe5 (code v.0608, 0610)

If no altimeter is present, it is the internal vacuum counts (redundant).

If an altimeter is present, use the exc_stat bit 0x4000 to decide if an ADP(=1) or Tritech(0).

If Tritech, then distance_to_bottom [m] = Alt*0.02.

If ADP, then

LSB = depth[m] to the bottom (80=not detected, 104 = NOT being used).

MSB = maximum intensity [counts] of the return signal

current and **bat** are the average pump current and battery voltage as sampled in the bottom turn.

Psurf [counts] gives the surface reference pressure for the sensor profile data. Psurf provides the reference counts to adjust the profile data so 0=surface.

pitch = absolute degrees target pitch to maintain during descent (-pitch) and ascent (+pitch).

head = [degrees TRUE] of the target heading to steer (TRUE instead of magnetic, start v.0608)

drx, dry [m] dead-reckoning distance-traveled. Starting v0608, includes integration in bottom-turn.

ydeg, dy, xdeg, dx = waypoint Latitude and Longitude values [decimal degrees].

n_badamp = # of times during the dive that the pump current exceeded the maximum current limit and caused the pump to be prematurely turned off (maximum current limit is in eeprom, and can be set via shore cmd 'C').

navg = # of points averaged for each profile output datum.

ti_pump :MSB = pump time s/10 required before Spray starts to become buoyant at the bottom turn.

LSB = pump time s/10 since becoming buoyant. This includes the time pumping at the surface.

The total pump time = (MSB + LSB)*10

vac [in-Hg*100], internal vacuum : value of -1000 = -10.00 in-Hg vacuum = nominal value.

idive is the profile dive number. This is required if multiple profiles are in the same SBD message.

The Spray writes out the sensor data, followed by the engineering data, for each profile. So when an engineering packet is found, the dive number should be parsed out and used to tag the preceeding sensor data.

miss_id is a mission identifier, set in the Spray EEPROM. Of the two bytes, three parameters are specified, as shown in the following c code,

```
m1 = ( miss_id & 0xff00 ) >>8; // MSB = deployment year (assumed range 0..99)
```

```
m2 = ( miss_id & 0x00f0 ) >> 4; // = month (assumed range = 1..12)
```

```
m3 = ( miss_id & 0x000f ); // = unique id (range 0..15)
```

max_amp = time and amps as recorded while pumping during the bottom turn.

MSB = Time s/10 when the maximum pump amps occurred.

LSB = Maximum Pump Current (amps = LSB*50 , constrained to be <256).

r_err =Integrated error in the heading PI loop, representing the roll bias required to head straight.

Roll Bias [degrees] = r_err /1.92.

t_SBD [s/10] = time that the SBD modem was actively trying to send a message during the last dive.

This is the accumulated time between sending a 'SEND_SBD' command and getting a reply back from the modem (of either success or failure), and is a measure of the actual transmit baud rate.

ntries = The number of connect tries attempted during the LAST surface communications.

nsent = The number of SBD messages sent during the LAST surface communications.

sbd_stat = the status of the last attempt to send a message. Values are:

- 0 = no SBD message queued in ISU to send.
- 1 = SBD message was successfully sent.
- 2 = an error occurred while attempting to send.
- 3 = parsing error while reading the command response.
- 8 = no satellite (<10 s for reply with error=2).
- 9 = timeout (no command response from ISU ...not good).

sbd_shore_stat is the status of parsing a shore command:

- 0 = there was no incoming SBD message reported by the modem.
- 1 = data packet received successfully.
- 2 = ISU received GS packet with an error, but continue anyways.
- 3 = ISU requests a re-transmit of the GS packet.
- 4 = ISU received wrong # of bytes or bad CRC.
- 5 = no 'X' character. found in the incoming message.
- (above)+5 = ISU detected a bad sub-command.

exc_stat = Exception status word. ..Let **b0 = bit[0] = Lsbit**, **b15 = bit[15] = Msbit**.

- | | |
|--|--|
| b0 = Pump Recovery was required. | b8 = CF1 Close File error. |
| b1 = Drop Weight Activated. | b9 = High amps at the 50 m pump. |
| b2 = Pressure>20 m 'at surface.' | b10= Press=0 (broken) 'at surface.' |
| b3 = Depth>1500 m detected. | b11= No SBD sent in the last 48 hrs. |
| b4 = Altimeter triggered turn-around. | b12= Cannot overcome the current. |
| b5 = Backed off the surface. | b13= Spurious Reset Detected. |
| b6 = Extra pumps req'd at end-ascent. | b14= Altimeter reading is from an ADP. |
| b7 = Took >700 s to leave the surface. | b15= Not Used. |

surf_tm = time spent at the surface, both before and after GPS/SBD comms.

MSB= time[s/10] at end-of-dive pumping extra oil, before turning on the GPS.

LSB = time[s/10] at start-of-dive to leave the surface (time to deflate the bladders).

Test pattern (ID=0xfn)

This is an incrementing pattern of modulo M between successive points (normally M=1).

Byte	Contents
0	0xfn : MSNibble= 0xf identifies it as a test packet. The LSNibble n is used to identify multiple test packets in the same SBD (normally set to 0).
1	Msbyte of NN (NN = number of bytes in this sensor packet).
2	Lsbyte of NN.
3	M = modulo used to create the test pattern.
4	Test[0] = 0;
J+4	Test[J] = Test[J-1] + M (rolls over when >255).
NN-1	';' = end packet delimiter

Sample checksum verification code

Given the definition of the SBD message (see p. 1), the following function should be called as:

```
Good = ISU_chk_crc( a[ ], nn+3 );
```

Where a[] points to the SBD message, so
 a[0] = 'X' = the first byte in the SBD, and
 nn = # data bytes

(nn = a[1]*256 + a[2] : nn does NOT include the first 3 bytes, and so the call requires nn+3 = total number of bytes in the SBD message).

```
short ISU_chk_crc(unsigned char a[], short n ) //*****
// returns 1 if correct, 0 if bad

{ unsigned char c0, c1;
  long x=0;
  short i, y;

  printf("chk_crc of %d bytes: \n",n);
  if (n<0) return(0); // safeguard for illegal value

  for (i=0; i<n; i++)
    x += a[i]; // add up bytes

  if ( (a[n] == '$' ) && (a[n+3] == '>') )
  { // then we see the right delimiters
    c1 = a[n+1] - 48; // range should be 0..15
    c0 = a[n+2] - 48; // range should be 0..15
    y = c1*16 + c0; //=chksum value encoded in packet
  }
  else
  { return(0); } // else not right delimiters, return false

  i = (x & 0xff); //LSByte of computed chksum, should equal y

  return( i==y );
} //*** end ISU_chk_crc *****
```