TAKING INVENTIVE STEPS IN INFRARED....

Premier Sensor Communications protocol

Issue Number: 1.44
Issue Date: 20/08/2015
Document: TDS0045
Firmware Version: Various

Dynament
Premier House
The Village
South Normanton
Derbyshire DE55 2DS
United Kingdom

AMENDMENT RECORD		Original date of issue: 10/01/2012	
Date	**Modification**	**New Issue**	**Approval Signature**
20/08/2015	Bug in Zero example section 1.6.2	1.44	F. Kups
21/05/2015	Footer information does not match the revision	1.43	F. Kups
28/01/2015	Fix Typo errors in Live data Version 7 – sections 1.5.4 CO2 & C3H8 swapped	1.42	F. Kups
19/12/2014	Fix Typo errors in CRC – sections 1.6.5, 1.6.6, 1.6.7, 1.6.8 & 1.6.9	1.41	F. Kups
09/12/2014	Added version numbers to read Serial number	1.40	F. Kups
23/10/2014	'C' example of CRC calculation added	1.39	F. Kups
13/08/2014	Fix typo in section 1.6.5, 2.0 should be 2.5	1.38	F. Kups
04/07/2014	Added example for read live data dual command (0x2C) Added example 2 for live data simple command (0x06) Removed change note from footer – N/A in this document	1.37	F. Kups
01/05/2014	Show decimal and hex numbers in the variable id's	1.36	F. Kups
11/03/2014	Add example for serial number read function	1.35	F. Kups
10/03/2014	Added example for spanning a single range gas sensor	1.34	F. Kups
08/01/2014	Details of status flags added to section 2.7	1.33	F. Kups
29/11/2013	Example data for negative readings using integers in the live data, section 2.6 Remove duplicated modification note 1.30 below. Added Approval Signature to all modification versions. Added ACK bytes to sensor response to write requests examples for clarification. Re-formatted the amendment Record table.	1.32	F. Kups
10/10/2013	Error fixed: 12 p2pNAKdeviceFault	1.31	F. Kups
20/05/2013	Match example data in 1.5.4.1 with command 0x01	1.30	F. Kups
14/05/2013	Error fixed in 1.5.4.1 RD incorrectly identified as 0x2D	1.29	F. Kups
07/05/2013	Added CRCs to examples	1.28	F. Kups
23/04/2013	Checksum corrected in example in section 1.6.8.1	1.27	F. Kups
11/04/2013	Live data structure 6 added, was missing Config structures 9 added Sections 1.5.4.1 & 1.6.8.1 – reference to Jig controller removed. Live data 2 added to section 1.4 Section 2.7 added to cover new status flags Some re-formatting to keep consistency throughout the document.	1.26	F. Kups
26/03/2013	Config structures 8 added Live data structures 6 added	1.25	F. Kups
29/10/2012	Structure file live data version 5 error fixed: Fields Reading and multiplier are in the wrong order Example added for integer readings conversion	1.24	F. Kups
18/08/2012	Structure file live data version 4 added Structure numbers removed to avoid confusion with structure version numbers	1.23	F. Kups
31/08/2012	Structure file showing incorrect version, section 2.2	1.22	F. Kups

Dynament
Premier House
The Village
South Normanton
Derbyshire DE55 2DS
United Kingdom

21/06/2012	Extra error codes added to section 1.5.	1.21	F. Kups
02/05/2012	Byte stuffing examples added.	1.20	F. Kups
20/04/2012	Added configuration data structure for triple range sensor. Added version numbers to config data structures.	1.19	F. Kups
14/03/2012	Added configuration data structure for Auto ranging sensor. Renamed other structures for clarity.	1.18	F. Kups
23/02/2012	Amendment Record added.	1.17	F. Kups
23/02/2012	Status flag : FLAG_WARM_UP changed back to FLAG_USER_CSUM To provide compatibility with existing sensor firmware Status flags 2: updated to include FLAG_WARM_UP		F. Kups

Dynament
Premier House
The Village
South Normanton
Derbyshire DE55 2DS
United Kingdom

Terms and conditions for the use of Premier Protocol

When using the **"Calibration-only"** version of the protocol, the user has chosen to accept full responsibility for any changes they make to the calibration of the sensor. This is a condition of sale imposed by the Insurers of Dynament Ltd.

This decision does not affect the warranty period against defects in materials or workmanship.

1 COMMUNICATIONS PROTOCOL

The communications protocol used by the Premier sensor is used for communications between devices connected via an RS232 connection. This point-to-point, P2P, protocol is a frame-based protocol.

1.1 Baud rates

Four baud rates are available: 4800 8N1, 9600 8N1, 19200 8N1 and 38400 8N1 (No parity, 1 stop bit).

1.2 Control Byte Constants

The following control byte constants are used in the P2P protocol[i].

Description	Command	Code	Binary
Read	RD	= 0x13	(00010011)
Data Link Escape	DLE	= 0x10	(00010000)
Write	WR	= 0x15	(00010101)
Acknowledge	ACK	= 0x16	(00010110)
Negative Acknowledge	NAK	= 0x19	(00011001)
Single Data Frame	DAT	= 0x1A	(00011010)
End of Frame	EOF	= 0x1F	(00011111)
Write Password 1	WP1	= 0xE5	(11100101)
Write Password 2	WP2	= 0xA2	(10100010)

1.3 Frame Structure

The start of a frame is indicated by a DLE byte followed by the type of frame to follow (RD, WR, ACK, NAK, DAT). The end of frame is indicated by a DLE byte followed by an EOF byte. The end of the frame is followed by either a simple checksum or a CRC based on the data.

Field	DLE	CMD	PAYLOAD	DLE	EOF	Check Sum	CRC
Octet	1	1	N	1	1	2	2

Note: Each of the constants has bit 4 set and so is slip-resistant (i.e. if shifted this bit will be out of position). The values have a Hamming Distance of 2 (each code is at least 2 bits different from every other code).

In information theory, the Hamming distance between two strings of equal length is the number of positions at which the corresponding symbols are different. Put another way, it measures the minimum number of substitutions required to change one string into the other, or the number of errors that transformed one string into the other.

The Hamming distance between:

- 1011101 and 1001001 is 2.
- 2173896 and 2233796 is 3.
- "toned" and "roses" is 3.

Any DLE bytes that occur between a frame's start and end are prefixed with another DLE (*byte-stuffing*). DLE stuffing / unstuffing mechanism should be considered.

Following the EOF is a 16-bit checksum or a 16-bit CRC of the entire frame,
Checksum - each byte is added to produce the checksum.

$$Checksum = \sum_{i=1}^{n} Byte_i \quad \text{(DLE through EOF)}$$

CRC - the CRC is based on:

Width = 16 bits
Polynomial = 0x8005

When any variable is adjusted by “WRITE” command, two bytes password (WP1 & WP2) shall follow the CMD byte to prevent unintended parameter change

1.4 Variables

Each piece of accessible data on a device is referred to as a *Variable*. Each variable is referenced by a *Variable ID*. A *variable ID* may be any number up to 255 bytes long.

The available Variables and their corresponding Variable IDs depend on the type of device, but here are a few examples for the Premier sensor:

Purpose	Variable id	Comments
Config data	0, (0x00)	read / write
Live Data	1, (0x01)	read only
Zero Sensor1	2, (0x02)	write only
Span Sensor1_R1	3, (0x03)*	write only
Live Data Simple	6, (0x06)	read only
User Data	11, (0x0B)	read / write
Zero Sensor2	22, (0x16)	write only
Span Sensor2	3, (0x03)*	write only
Span Sensor1_R2	3, (0x03)*	write only
Live Data 2	44, (0x2C)	read only
Read Serial Number	48, (0x30)**	read only

The dual sensor has 2 detectors, sensor1 and sensor2. Sensor1 has 2 ranges R1 (CH4) and R2 (C3H8).

The structure of the data returned in each variable usually depends both on the type of device and the version of firmware running on the device.
Refer to section 0 for more information.

* The dual sensor requires an additional parameter for the span command see section 2.5
** This is only available on certain versions, see section 1.5.5

1.5 Reading a Variable

Send a read frame with the Variable ID to be read:

DLE	RD	PAYLOAD	DLE	EOF	Csum hi	Csum lo
		Variable ID				

Device response on success, where requested variable data < 255 bytes:

DLE	DAT	PAYLOAD		DLE	EOF	Csum hi	Csum lo
		DATA_LEN	DATA				

Note: DATA_LEN field indicates the length of data field only.

Device response on failure:

DLE	NAK	reason

Where ‘reason’ is a single byte failure code, the meaning of which depends on the device type, i.e.

1 p2pNAKvarNotReadable,
2 p2pNAKvarNotWritable,
3 p2pNAKoutOfRange,
4 p2pNAKincorrectLength,
5 p2pNAKunexpectedBytes,
6 p2pNAKchecksumFailed,
7 p2pNAKincorrectVersion,
8 p2pNAKbusy,
9 p2pNAKinvalidData,
10 p2pNAKinvalidState,
11 p2pNAKserialError,
12 p2pNAKdeviceFault

1.5.1 Read live data

Send the following bytes:

DLE, RD, Variable ID, DLE, EOF, Checksum High byte, Checksum low byte or CRC High byte, CRC low byte i.e.
0x10, 0x13, 0x01, 0x10, 0x1F, **0x00, 0x53** or **0x1B, 0xD0**

Device response on success:
DLE, DAT, Data length, Data, DLE, EOF, **Checksum High byte, Checksum low byte** or **CRC High byte, CRC low byte**, i.e.

0x10	DLE
0x1A	DAT
0x14	Data length
0x01, 0x00	Version
0x00, 0x00	Status flags
0x00, 0x00, 0x28, 0x41	Gas reading, 32 bit floating point - IEEE-754 format
0x00, 0x00, 0x1E, 0x42	Temperature, 32 bit floating point - IEEE-754 format
0x2C, 0x04	Detector signal
0x86, 0x02	Reference signal
0x80, 0x1A, 0x09, 0xBC	Absorbance, 32 bit floating point - IEEE-754 format
0x10	DLE
0x1F	EOF
0x03 or **0x0F**	**Checksum high byte** or **CRC high byte**
0xA5 or **0xDB**	**Checksum low byte** or **CRC high byte**

Note 1: 0x41280000 = 10.50
Note 2: the data length may increase depending upon the sensor firmware.

The data can be read and any extra bytes can be ignored if not needed.

Note 3: if the data, bytes after the data length and before the DLE character contains any 0x10 bytes then every occurrence will be preceded with another 0x10 byte. This is termed byte stuffing. These extra bytes should be used in calculating the checksum but discarded and not used for data or data length.

1.5.2 Read live data simple

Send the following bytes:

DLE, RD, Variable ID, DLE, EOF, Checksum High byte, Checksum low byte or CRC High byte, CRC low byte, i.e.
0x10, 0x13, 0x06, 0x10, 0x1F, **0x00, 0x58** or **09, 0xBF**

Device response on success:
DLE, DAT, Data length, Data, DLE, EOF, **Checksum High byte, Checksum low byte** or **CRC High byte, CRC low byte**, i.e.

1.5.2.1 Example 1 (version 1)

0x10	DLE
0x1A	DAT
0x08	Data length
0x01, 0x00	Version
0x00, 0x00	Status flags
0x00, 0x00, 0x60, 0x40	Gas reading, 32 bit floating point - IEEE-754 format (3.5)
0x10	DLE
0x1F	EOF
0x01 or **0x53**	**Checksum high byte** or **CRC high byte**
0x02 or **0xAE**	**Checksum low byte** or **CRC high byte**

Note 1: 0x40600000 = 3.50

1.5.2.2 Example 2 (version 4)

0x10	DLE
0x1A	DAT
0x08	Data length
0x04, 0x00	Version
0x00, 0x00	Status flags
0XA4, 0x70, 0xBD, 0x3F	Gas reading, 32 bit floating point - IEEE-754 format
0x10	DLE
0x1F	EOF
0x02 or **0xE7**	**Checksum high byte** or **CRC high byte**
0x75 or **0xE8**	**Checksum low byte** or **CRC high byte**

Note 2: 0x3FBD70A4 = 1.48

1.5.3 Read live data Dual sensor (structure 4, version 3)

The following example is for the Dual sensor with Version 3 live data structure

Send the following bytes:

DLE, RD, Variable ID, DLE, EOF, Checksum High byte, Checksum low byte or CRC High byte, CRC low byte i.e.
0x10, 0x13, 0x01, 0x10, 0x1F, **0x00, 0x53** or **0x1B, 0xD0**

Device response on success:
DLE, DAT, Data length, Data, DLE, EOF, **Checksum High byte, Checksum low byte** or **CRC High byte, CRC low byte**, i.e.

0x10	DLE
0x1A	DAT
0x2E	Data length
0x03, 0x00	Version
0x00, 0x00	Status flags
0xAE, 0x47, 0x61, 0x3E	Gas reading 1 (CH4), 32 bit floating point - IEEE-754 format
0x00, 0x00, 0xAC, 0x41	Temperature, 32 bit floating point - IEEE-754 format
0XB8, 0x1E, 0x05, 0x3E	Gas reading 2 (CO2), 32 bit floating point - IEEE-754 format
0x66, 0x01, 0XD4, 0x44	Detector 1 signal
0XD6, 0x88, 0x53, 0x44	Reference signal
0x8F, 0XC2, 0x75, 0x3C	Absorbance (Fa1), 32 bit floating point - IEEE-754 format
0x1C, 0x1F, 0x01, 0x00	Sensor powered time (divide by 100 to get seconds)
0x6B, 0xFA, 0x72, 0x44	Detector 2 signal
0x30, 0x4C, 0XA6, 0x3C	Absorbance (Fa2), 32 bit floating point - IEEE-754 format
0x00, 0x00	Status flags 2
0x8F, 0xC2, 0XF5, 0x3C	Gas reading 3 (Propane), 32 bit floating point - IEEE-754 format
0x10	DLE
0x1F	EOF
0x0B or **0xBD**	**Checksum high byte** or **CRC high byte**
0xCC or **0x18**	**Checksum low byte** or **CRC high byte**

Note 1:

Reading 1	0x3E6147AE	= 0.22 %v/v CH4
Temperature	0X41AC000	= 21.5 degC
Reading 2	0x3E051EB8	= 0.13 %v/v CO2
Fa1	0x3C75C28F	= 0.015
upTime	0x00011F1C	= 73500 (12 Minutes 15 Seconds)
Fa2	0x3CA64C30	= 0.0203
Reading 3	0x3CF5C28F	= 0.03 %v/v C3H8

Note2: The detector signals have changed from integers to floats

Detector1	0x44D40166	= 1696.04
Reference	0x445388D6	= 846.1381
Detector2	0x4472FA6B	= 971.9128

1.5.4 Read live data Dual sensor (Version 7)

The following example is for the Dual sensor with Version 7 live data structure.
This version caters for both the single and dual gas type sensors. The gas reading for any unused range will return typically -250% FSD), this is -12.5 for a 5% range and -250 for a 100% range.

Send the following bytes:

DLE, RD, Variable ID, DLE, EOF, Checksum High byte, Checksum low byte or CRC High byte, CRC low byte i.e.
0x10, 0x13, 0x2C, 0x10, 0x1F, **0x00, 0x7E** or **0x19, 0xB4**

Device response on success:
DLE, DAT, Data length, Data, DLE, EOF, **Checksum High byte, Checksum low byte** or **CRC High byte, CRC low byte**, i.e.

0x10	DLE
0x1A	DAT
0x32	Data length
0x07, 0x00	Version
0x00, 0x00	Status flags
0x8F 0xC2 0x75 0x3D	Gas reading 1 (CH4), 32 bit floating point - IEEE-754 format
0xCE 0x97 0x06 0x42	Temperature, 32 bit floating point - IEEE-754 format
0x0A 0xD7 0xA3 0x3D	Gas reading 2 (CO2), 32 bit floating point - IEEE-754 format
0x94 0xA9 0x0F 0x45	Detector 1 signal, 32 bit floating point - IEEE-754 format
0xE2 0x7C 0x93 0x44	Reference signal, 32 bit floating point - IEEE-754 format
0x80 0xAA 0xFB 0x3B	Absorbance (Fa1), 32 bit floating point - IEEE-754 format
0x7C 0x63 0x04 0x00	Sensor powered time (divide by 100 to get seconds)
0x11 0x89 0x3D 0x44	Detector 2 signal, 32 bit floating point - IEEE-754 format
0x50 0xC4 0x20 0x3D	Absorbance (Fa2), 32 bit floating point - IEEE-754 format
0x00 0x00	Status flags 2
0x29 0x5C 0x8F 0x3D	Gas reading 3 (C3H8), 32 bit floating point - IEEE-754 format
0x00 0x00	Status flags 3
0xFF 0xFF	Status flags 4
0x10	DLE
0x1F	EOF
0x13 or **0x75**	**Checksum high byte** or **CRC high byte**
0x65 or **0x5F**	**Checksum low byte** or **CRC high byte**

Note 1:	Reading 1	0x3D75C28F	= 0.06 %v/v CH4
	Temperature	0X420697CE	= 33.6 degC
	Reading 2	0x3DA3D70A	= 0.08 %v/v CO2
	Fa1	0x3BFBAA80	= 0.007680237
	upTime	0x0004637C	= 287612 (47mins 56 secs)
	Fa2	0x3D20C450	= 0.03924972
	Reading 3	0x3D8F5C29	= 0.07 %v/v C3H8

1.5.5 Read Serial Number

Send the following bytes:

DLE, RD, Variable ID, DLE, EOF, Checksum High byte, Checksum low byte or CRC High byte, CRC low byte i.e.
0x10, 0x13, 0x30, 0x10, 0x1F, **0x00, 0x82** or **0x98, 0x07**

Device response on success:
DLE, DAT, Data length, Data, DLE, EOF, **Checksum High byte, Checksum low byte** or **CRC High byte, CRC low byte**, i.e.

0x10	DLE
0x1A	DAT
0x06	Data length
0x31	1
0x32	2
0x33	3
0x34	4
0x35	5
0x36	6
0x10	DLE
0x1F	EOF
0x01 or **0x47**	**Checksum high byte** or **CRC high byte**
0x94 or **0xBC**	**Checksum low byte** or **CRC high byte**

Note: This feature is only available on Firmware V07.17.06 onwards
Note This feature is NOT available on SIL Firmware V07.17.00

1.5.6 Byte Stuffing Read Data

If a DLE character exists in the data, which is highly possible, then the extra DLE character that is inserted immediately after the first DLE is not used, however, it is used in the checksum calculations.

1.5.6.1 Byte Stuffing Read Live Data Example

Send the following bytes:

DLE, RD, Variable ID, DLE, EOF, Checksum High byte, Checksum low byte or CRC High byte, CRC low byte i.e.
0x10, 0x13, 0x01, 0x10, 0x1F, **0x00, 0x53** or **0x1B, 0xD0**

A typical device response on success:

DLE, DAT, Data length, Data, DLE, EOF, **Checksum High byte, Checksum low byte** or **CRC High byte, CRC low byte**, i.e.

0x10	DLE
0x1A	DAT
0x2E	Data length
0x03, 0x00	Version
0x00, 0x00	Status flags
0xAE, 0x47, 0x61, 0x3E	Gas reading 1 (CH4), 32 bit floating point - IEEE-754 format
0x10, 0X10, 0x00, 0xAC, 0x41	Temperature, extra DLE to be ignored
0XB8, 0x1E, 0x10, 0X10, 0x3E	Gas reading 2, extra DLE to be ignored
0x66, 0x01, 0XD4, 0x44	Detector 1 signal
0XD6, 0x88, 0x53, 0x44	Reference signal
0x8F, 0XC2, 0x75, 0x3C	Absorbance (Fa1), 32 bit floating point - IEEE-754 format
0x1C, 0x1F, 0x01, 0x00	Sensor powered time (divide by 100 to get seconds)
0x6B, 0xFA, 0x72, 0x44	Detector 2 signal
0x30, 0x4C, 0XA6, 0x3C	Absorbance (Fa2), 32 bit floating point - IEEE-754 format
0x00, 0x00	Status flags 2
0x8F, 0xC2, 0XF5, 0x3C	Gas reading 3 (Propane), 32 bit floating point - IEEE-754 format
0x10	DLE
0x1F	EOF
0x0C or **0xF5**	**Checksum high byte** or **CRC high byte**
0x0C or **0xFA**	**Checksum low byte** or **CRC high byte**

Note 1:

Reading 1	0x3E6147AE	= 0.22 %v/v CH4
Temperature	0X41AC010	= 21.5 degC
Reading 2	0x3E101EB8	= 0.1407 %v/v CO2
Fa1	0x3C75C28F	= 0.015
upTime	0x00011F1C	= 73500 (12 Minutes 15 Seconds)
Fa2	0x3CA64C30	= 0.0203
Reading 3	0x3CF5C28F	= 0.03 %v/v C3H8

Note2: The detector signals have changed from integers to floats

Detector1	0x44D40166	= 1696.04
Reference	0x445388D6	= 846.1381
Detector2	0x4472FA6B	= 971.9128

Note 3: The Data length is the actual number of valid data bytes. The number of bytes received is increased by 2 due to byte stuffing.

1.6 Writing a Variable

Send a write frame with the Variable ID to be written:

DLE	WR	PAYLOAD			DLE	EOF	Csum hi	Csum lo
		WP1	WP2	Variable ID				

Device response on success:

DLE	ACK

Where data to write is < 255 bytes, send a DAT frame:

DLE	DAT	PAYLOAD		DLE	EOF	Csum hi	Csum lo
		DATA_LEN	DATA				

Device response on write success:

DLE	ACK

Device response on write failure:

DLE	NAK	reason

Where 'reason' is a single byte failure code, the meaning of which depends on the device type, i.e.

Reason = 1, NotWritable
Reason = 2, WriteOutOfRange
Reason = 3, BadDataLength
Reason = 4, IncorrectVersion

1.6.1 Zero sensor 1

Send the following bytes:

DLE, WR, WP1, WP2, Variable ID, DLE, EOF, Checksum High byte, Checksum low byte or **CRC High byte, CRC low byte,**
DLE, DAT, Data Len, Data, DLE, EOF, Checksum High byte, Checksum low byte or **CRC High byte, CRC low byte**

Send :
0x10, 0x15, 0xE5, 0XA2, 0x02, 0x10, 0x1F, **0x01, 0xDD** or **0xED, 0xD6**

Device response on success:
0x10 DLE
0x16 ACK

Send:
0x10, 0x1A, 0x00, 0x10, 0x1F, **0x00, 0x59** or **0x2F, 0xC7**

Device response on success:
0x10 DLE
0x16 ACK

Note: in this case there is no data, assumed value of zero.
This is the HC channel in dual sensors

1.6.2 Zero sensor 2 (dual sensor)

Send the following bytes:

DLE, WR, WP1, WP2, Variable ID, DLE, EOF, Checksum High byte, Checksum low byte or **CRC High byte, CRC low byte, DLE, DAT, Data Len, Data, DLE, EOF, Checksum High byte, Checksum low byte** or **CRC High byte, CRC low byte**

Send:
0x10, 0x15, 0xE5, 0XA2, 0x16, 0x10, 0x1F, **0x01, 0xF1** or **0xEC, 0xC6**

Device response on success:
0x10 DLE
0x16 ACK

Send:
0x10, 0x1A, 0x00, 0x10, 0x1F, **0x00, 0x59** or **0x2F, 0xC7**

Device response on success:
0x10 DLE
0x16 ACK

Note: in this case there is no data, assumed value of zero.
This is the CO2 channel

1.6.3 Span sensor, 2.5% gas (single range sensor)

Send the following bytes:

DLE, WR, WP1, WP2, Variable ID, DLE, EOF, Checksum High byte, Checksum low byte or **CRC High byte, CRC low byte, DLE, DAT, Data Len, Data, DLE, EOF, Checksum High byte, Checksum low byte** or **CRC High byte, CRC low byte**
i.e.

Send:
0x10, 0x15, 0xE5, 0XA2, 0x03, 0x10, 0x1F, **0x01, 0xDE** or **0x6D, 0xC1**

Device response on success:
0x10 DLE
0x16 ACK

Send:
0x10, 0x1A, 0x04, 0x00, 0x00, 0x20, 0x40, 0x10, 0x1F, **0x00, 0xBD** or **0x27, 0x54**

Where 0x00, 0x00, 0x20, 0x40 = 2.5, IEEE float lsb first.

Device response on success:
0x10 DLE
0x16 ACK

1.6.4 Span sensor, 20000ppm gas (single range sensor)

Send the following bytes:

DLE, WR, WP1, WP2, Variable ID, DLE, EOF, Checksum High byte, Checksum low byte or **CRC High byte, CRC low byte, DLE, DAT, Data Len, Data, DLE, EOF, Checksum High byte, Checksum low byte** or **CRC High byte, CRC low byte**
i.e.

Send:
0x10, 0x15, 0xE5, 0XA2, 0x03, 0x10, 0x1F, **0x01, 0xDE** or **0x6D, 0xC1**

Device response on success:
0x10 DLE
0x16 ACK

Send:
0x10, 0x1A, 0x04, 0x00, 0x40, 0x9C, 0x46, 0x10, 0x1F, **0x01, 0x7F** or **0x09, 0x1A**

Where 0x00, 0x40, 0x9C, 0x46 = 20000 IEEE float lsb first.

Device response on success:
0x10 DLE
0x16 ACK

1.6.5 Span sensor range 0, 2.5% gas (dual sensor CH4L)

The dual sensor uses the Dual gas sensor data structure

Send the following bytes:

DLE, WR, WP1, WP2, Variable ID, DLE, EOF, Checksum High byte, Checksum low byte or **CRC High byte, CRC low byte, DLE, DAT, Data Len, Data, DLE, EOF, Checksum High byte, Checksum low byte** or **CRC High byte, CRC low byte**
i.e.

Send:
0x10, 0x15, 0xE5, 0XA2, 0x03, 0x10, 0x1F, **0x01, 0xDE** or **0x6D, 0xC1**

Device response on success:
0x10 DLE
0x16 ACK

Send:
0x10, 0x1A, 0x06, 0x00, 0x00, 0x20, 0x40, 0x00, 0x00, 0x10, 0x1F, **0x00, 0xBF** or **0x7E, 0xF3**

Where: 0x00, 0x00, 0x20, 0x40 = 2.5, IEEE float lsb first.
0x00, 0x00 = Range 0, CH4L

Device response on success:
0x10 DLE
0x16 ACK

1.6.6 Span sensor range 1, 99.5% gas (dual sensor CH4H)

Send the following bytes:

DLE, WR, WP1, WP2, Variable ID, DLE, EOF, Checksum High byte, Checksum low byte or **CRC High byte, CRC low byte, DLE, DAT, Data Len, Data, DLE, EOF, Checksum High byte, Checksum low byte** or **CRC High byte, CRC low byte**
i.e.

Send:
0x10, 0x15, 0xE5, 0XA2, 0x03, 0x10, 0x1F, **0x01, 0xDE** or **0x6D, 0xC1**

Device response on success:
0x10 DLE
0x16 ACK

Send:
0x10, 0x1A, 0x06, 0x00, 0x00, 0xC7, 0x42, 0x01, 0x00, 0x10, 0x1F, 0x01, 0x69

Where: 0x00, 0x00, 0xC7, 0x42 = 99.5, IEEE float lsb first.
0x01, 0x00 = Range 1, CH4H

Device response on success:
0x10 DLE
0x16 ACK

1.6.7 Span sensor range 2, 1.1% gas (dual sensor C3H8)

Send the following bytes:

DLE, WR, WP1, WP2, Variable ID, DLE, EOF, Checksum High byte, Checksum low byte or **CRC High byte, CRC low byte, DLE, DAT, Data Len, Data, DLE, EOF, Checksum High byte, Checksum low byte** or **CRC High byte, CRC low byte**
i.e.

Send:
0x10, 0x15, 0xE5, 0XA2, 0x03, 0x10, 0x1F, **0x01, 0xDE** or **0x6D, 0xC1**

Device response on success:
0x10 DLE
0x16 ACK

Send:
0x10, 0x1A, 0x06, 0xCD, 0xCC, 0x8C, 0x3F, 0x02, 0x00, 0x10, 0x1F, **0x02, 0xC5** or **0xF0, 0x38**

Where: 0xCD, 0xCC, 0x8C, 0x3F = 1.1, IEEE float lsb first.
0x02, 0x00 = Range 2, C3H8

Device response on success:
0x10 DLE
0x16 ACK

1.6.8 Span sensor range 3, 2.0% gas (dual sensor CO2)

Send the following bytes:

DLE, WR, WP1, WP2, Variable ID, DLE, EOF, Checksum High byte, Checksum low byte or **CRC High byte, CRC low byte, DLE, DAT, Data Len, Data, DLE, EOF, Checksum High byte, Checksum low byte** or **CRC High byte, CRC low byte**
i.e.

Send:
0x10, 0x15, 0xE5, 0XA2, 0x03, 0x10, 0x1F, **0x01, 0xDE** or **0x6D, 0xC1**

Device response on success:
0x10 DLE
0x16 ACK

0x10, 0x1A, 0x06, 0x00, 0x00, 0x00, 0x40, 0x03, 0x00, 0x10, 0x1F, **0x00, 0xA2** or **0xC2, 0xD2**

Where: 0x00, 0x00, 0x00, 0x40 = 2.0, IEEE float lsb first.
0x03, 0x00 = Range 3, CO2

Device response on success:
0x10 DLE
0x16 ACK

1.6.9 Byte Stuffing Write Data

1.6.9.1 Span Sensor range 0, 2.25% gas Example

Send the following bytes:

DLE, WR, WP1, WP2, Variable ID, DLE, EOF, Checksum High byte, Checksum low byte or **CRC High byte, CRC low byte, DLE, DAT, Data Len, Data, DLE, EOF, Checksum High byte, Checksum low byte** or **CRC High byte, CRC low byte**
i.e.
Send
0x10, 0x15, 0xE5, 0XA2, 0x03, 0x10, 0x1F, **0x01, 0xDE** or **0x6D, 0xC1**

Device response on success:
0x10 DLE
0x16 ACK

Send
0x10, 0x1A, 0x06, 0x00, 0x00, 0x10, 0x10, 0x40, 0x00, 0x00, 0x10, 0x1F, **0x00, 0xBF** or **0x7C, 0x50**

Where: 0x40100000 = 2.25, IEEE float lsb first.
0x0000 = Range 0, CH4L

Device response on success:
0x10 DLE
0x16 ACK

Note 1: The Data length remains at 6, however, 7 bytes are transmitted due to byte stuffing, the DLE character is added to the data stream and used by the checksum

1.6.10 CRC calculation

The CRC uses the polynomial 0x8005.

A 'C' example is as follows:

```
//define the CRC lookup table
static const unsigned short CRCTab[256] = {
      0x0000, 0x8005, 0x800f, 0x000a, 0x801b, 0x001e, 0x0014, 0x8011,
      0x8033, 0x0036, 0x003c, 0x8039, 0x0028, 0x802d, 0x8027, 0x0022,
      0x8063, 0x0066, 0x006c, 0x8069, 0x0078, 0x807d, 0x8077, 0x0072,
      0x0050, 0x8055, 0x805f, 0x005a, 0x804b, 0x004e, 0x0044, 0x8041,
      0x80c3, 0x00c6, 0x00cc, 0x80c9, 0x00d8, 0x80dd, 0x80d7, 0x00d2,
      0x00f0, 0x80f5, 0x80ff, 0x00fa, 0x80eb, 0x00ee, 0x00e4, 0x80e1,
      0x00a0, 0x80a5, 0x80af, 0x00aa, 0x80bb, 0x00be, 0x00b4, 0x80b1,
      0x8093, 0x0096, 0x009c, 0x8099, 0x0088, 0x808d, 0x8087, 0x0082,
      0x8183, 0x0186, 0x018c, 0x8189, 0x0198, 0x819d, 0x8197, 0x0192,
      0x01b0, 0x81b5, 0x81bf, 0x01ba, 0x81ab, 0x01ae, 0x01a4, 0x81a1,
      0x01e0, 0x81e5, 0x81ef, 0x01ea, 0x81fb, 0x01fe, 0x01f4, 0x81f1,
      0x81d3, 0x01d6, 0x01dc, 0x81d9, 0x01c8, 0x81cd, 0x81c7, 0x01c2,
      0x0140, 0x8145, 0x814f, 0x014a, 0x815b, 0x015e, 0x0154, 0x8151,
      0x8173, 0x0176, 0x017c, 0x8179, 0x0168, 0x816d, 0x8167, 0x0162,
      0x8123, 0x0126, 0x012c, 0x8129, 0x0138, 0x813d, 0x8137, 0x0132,
      0x0110, 0x8115, 0x811f, 0x011a, 0x810b, 0x010e, 0x0104, 0x8101,
      0x8303, 0x0306, 0x030c, 0x8309, 0x0318, 0x831d, 0x8317, 0x0312,
      0x0330, 0x8335, 0x833f, 0x033a, 0x832b, 0x032e, 0x0324, 0x8321,
      0x0360, 0x8365, 0x836f, 0x036a, 0x837b, 0x037e, 0x0374, 0x8371,
      0x8353, 0x0356, 0x035c, 0x8359, 0x0348, 0x834d, 0x8347, 0x0342,
      0x03c0, 0x83c5, 0x83cf, 0x03ca, 0x83db, 0x03de, 0x03d4, 0x83d1,
      0x83f3, 0x03f6, 0x03fc, 0x83f9, 0x03e8, 0x83ed, 0x83e7, 0x03e2,
      0x83a3, 0x03a6, 0x03ac, 0x83a9, 0x03b8, 0x83bd, 0x83b7, 0x03b2,
      0x0390, 0x8395, 0x839f, 0x039a, 0x838b, 0x038e, 0x0384, 0x8381,
      0x0280, 0x8285, 0x828f, 0x028a, 0x829b, 0x029e, 0x0294, 0x8291,
      0x82b3, 0x02b6, 0x02bc, 0x82b9, 0x02a8, 0x82ad, 0x82a7, 0x02a2,
      0x82e3, 0x02e6, 0x02ec, 0x82e9, 0x02f8, 0x82fd, 0x82f7, 0x02f2,
      0x02d0, 0x82d5, 0x82df, 0x02da, 0x82cb, 0x02ce, 0x02c4, 0x82c1,
      0x8243, 0x0246, 0x024c, 0x8249, 0x0258, 0x825d, 0x8257, 0x0252,
      0x0270, 0x8275, 0x827f, 0x027a, 0x826b, 0x026e, 0x0264, 0x8261,
      0x0220, 0x8225, 0x822f, 0x022a, 0x823b, 0x023e, 0x0234, 0x8231,
      0x8213, 0x0216, 0x021c, 0x8219, 0x0208, 0x820d, 0x8207, 0x0202
   };

#define addToChecksum(c, b) c = (c << 8) ^ CRCTab[(c >> 8) ^ b]

//initialise the checksum to zero before calculating the CRC
checksum = 0;
//use the following code for every byte in the stream of data, including the byte stuffing variables.
addToChecksum(checksum, byte);

//Send the checksum data as two bytes, MSB first
TxByte(checksum >> 8);
TxByte(checksum & 0xff);
```

2 VARIABLE STRUCTURES

The data transferred is byte orientated with the following sizes:

Byte	8 bits
Integer	2 bytes
Double	4 bytes
Long	4 bytes

2.1 Configuration data structure – Single Range Gas Sensor

```
struct {
        unsigned int Version;
        unsigned char SensorType[8];
        unsigned int ModeBits;              // bit 0 - voltage out '0', bridge out '1'
                                            // bit 1 – range 1 enable '1'
        unsigned int SensorFsd;
        double ZeroOffset;
        double ZeroCalTemperature;
        double SpanCalTemperature;
        double DacZero;
        double DacFsd;
        double PosZeroSuppress;
        double NegZeroSuppress;
        double CalibrationGasValue;
        double SpanOffset;
        double EI;
        double Power;
        unsigned char SerialNumber[10];
        double Rounding;
        unsigned int DacPowerup;
        unsigned int BaudRate;
        unsigned int WarmUpTime;
} ConfigData;
```

2.2 Configuration data structure– Dual Range Gas Sensor

```
struct {
            unsigned int Version;           // 4
            unsigned char SensorType[8];
            unsigned int ModeBits;
                  // bit 0                  AnalogueType
                  // bit 1                  Range1Flag
                  // bit 2                  Range2Flag
                  // bit 3                  Range3Flag
                  // bit 4                  FSDOverrangeFlag
                  // bit 5-8                PreFilter
                  // bit 9                  Enable dac monitor
                  // bit 10-11              Clamp, set bit to clamp reading to 00 = 200%, 01 = 150%, 10 = 125%, 11 = 100%
            unsigned int SensorFsd[2];
            double ZeroOffset;
            double ZeroCalTemperature;
            double SpanCalTemperature[2];
            double DacZero;
            double DacFsd;
            double PosZeroSuppress;
            double NegZeroSuppress;
            double TargetValue[2];
            double SpanOffset[2];
            double EI[2];
            double Power[2];
            unsigned char SerialNumber[10];
            double Rounding1;
            unsigned int DacPowerup;
            unsigned int BaudRate;
            unsigned int WarmUpTime;
            double TempCompPlus;
            double TempCompMinus;
            double Rounding2;
} ConfigData;
```

2.3 Configuration data structure– Triple Range Gas Sensor

```
struct {
        uint16_t uiVersion;                     // 7
        uint8_t aucSensorType[8];
        uint16_t uiModeBits;
            // bit 0            AnalogueType
            // bit 1            Range1Enable
            // bit 2            Range2Enable
            // bit 3            Range3Enable
            // bit 4            Range4Enable
            // bit 5-7          PreFilter
            // bit 8            Span gas check enable
            // bit 9            DacMOn enable
            // bit 10-11        Clamp, set bit to clamp reading to 00 = 200%, 01 = 150%, 10 = 125%, 11 = 100%
            // bit 12           DET1 reading format, 0 = %v/v 1 = %fsd
            // bit 13           DET2 reading format, 0 = %v/v 1 = %fsd
            // bit 14           liveData reading type, 0 = float, 1 = int
            // bit 15           Single channel emulation of calibration

        // 3 ranges
        double SensorFsd[3];                    // CH4L, CH4H, Propane
        double ZeroOffset;                      //
        double ZeroCalTemperature;   //
        double SpanCalTemperature[3];           // CH4L, CH4H, Propane
        double PosZeroSuppress;                 //
        double NegZeroSuppress;                 //
        double CalGas[3];                       // CH4L, CH4H, Propane
        double SpanOffset[3];           // CH4L, CH4H, Propane
        double El[3];                           // CH4L, CH4H, Propane
        double Power[3];                        // CH4L, CH4H, Propane
        double Rounding[3];                     // CH4L, CH4H, Propane
        double Filter0;                         // Most filtering
        double Filter1;                         //
        double Filter2;                         //
        double Filter3;                         //
        double Filter4;                         // Least filtering
        double FilterChangeHighTemp;            // Above 60 degC
        double FilterChangeGas;                 //
        double FilterChange;                    //
        uint16_t uiBaudRate;                    // comms speed
        uint16_t uiWarmUpTime;                  // seconds
        double TemperatureOffset;               // analogue temperature IC
        double DacZero;
        double DacFsd;
        uint16_t uiDacPowerup;double Rounding2;
} ConfigData;
```

2.4 Configuration data structure - Dual Gas Sensor

```
struct {
        unsigned int Version;                   // Version 6
        unsigned char SensorType[8];
        unsigned int ModeBits;
                // bit 0        Not used
                // bit 1        Range1Enable
                // bit 2        Range2Enable
                // bit 3        Range3Enable
                // bit 4        Range4Enable
                // bit 5-8      PreFilter
                // bit 9        Enhance reading
                // bit 10-11    Clamp, set bit to clamp reading to 00 = 200%, 01 = 150%, 10 = 125%, 11 = 100%
                // bit 12       DET1 reading format, 0 = %v/v 1 = %fsd
                // bit 13       DET2 reading format, 0 = %v/v 1 = %fsd
                // bit 14       AutoRange1
                // bit 15       AutoRange2

        // 3 ranges
        unsigned int SensorFsd[4];              // CH4L, CH4H, Propane, CO2
        double ZeroOffset[2];                   // HC, CO2 detectors
        double ZeroCalTemperature[2];           // HC, CO2 detectors
        double SpanCalTemperature[4];           // CH4L, CH4H, Propane, CO2
        double PosZeroSuppress[2];              // HC, CO2 detectors
        double NegZeroSuppress[2];              // HC, CO2 detectors
        double CalGas[4];                       // CH4L, CH4H, Propane, CO2
        double SpanOffset[4];                   // CH4L, CH4H, Propane, CO2
        double El[4];                           // CH4L, CH4H, Propane, CO2
        double Power[4];                        // CH4L, CH4H, Propane, CO2
        double Rounding[4];                     // CH4L, CH4H, Propane, CO2
        double TempCompPlus[4];                 // not used
        double TempCompMinus[4];                // not used
        unsigned int BaudRate;                  // comms speed
        unsigned int WarmUpTime;                // seconds
        double TemperatureOffset;               // analogue temperature IC

} ConfigData
```

```
struct
{
	uint16_t uiVersion;				// version 8
	uint8_t aucSensorType[8];
	uint16_t uiModeBits;
		// bit 0	AnalogueType
		// bit 1	Range1Enable
		// bit 2	Range2Enable
		// bit 3	Range3Enable
		// bit 4	Range4Enable
		// bit 5-7	PreFilter
		// bit 8	Span gas check enable
		// bit 9	DacMOn enable
		// bit 10-11	Clamp, set bit to clamp reading to 00 = 200%, 01 = 150%, 10 = 125%, 11 = 100%
		// bit 12	DET1 reading format, 0 = %v/v 1 = %fsd
		// bit 13	DET2 reading format, 0 = %v/v 1 = %fsd
		// bit 14	liveData reading type, 0 = float, 1 = int
		// bit 15	Single channel emulation of calibration

	// 3 ranges
	double SensorFsd[4];			// CH4L, CH4H, Propane, CO2
	double ZeroOffset[2];			//
	double ZeroCalTemperature[2];		//
	double SpanCalTemperature[4];		// CH4L, CH4H, Propane, CO2
	double PosZeroSuppress[2];		//
	double NegZeroSuppress[2];		//
	double CalGas[4];			// CH4L, CH4H, Propane, CO2
	double SpanOffset[4];			// CH4L, CH4H, Propane, CO2
	double El[4];				// CH4L, CH4H, Propane, CO2
	double Power[4];			// CH4L, CH4H, Propane, CO2
	double Rounding[4];			// CH4L, CH4H, Propane, CO2
	uint16_t uiBaudRate;			// comms speed
	uint16_t uiWarmUpTime;			// seconds
	double TemperatureOffset;		// analogue temperature IC
	double DacZero;
	double DacFsd;
	uint16_t uiDacPowerup;
	double SlopeBiogas;			// CO2 compensation
	double CH4CrossRefFactor;		// CH4 factor for other gases on CH4 scale
	double C3H8CrossRefFactor;		// C3H8 factor for other gases on C3H8 scale
} ConfigData;
```

2.5 Configuration data structure – Triple Range Gas Sensor with pressure compensation

Provisional data – under development

```
struct {
   unsigned int Version;                        // Version 9
   unsigned char SensorType[8];
   unsigned int ModeBits;
            // bit 0       Pressure enabe ('1') / disable ('0')
            // bit 1       Range1Enable
            // bit 2       Range2Enable
            // bit 3       Range3Enable
            // bit 4       Range4Enable
            // bit 5-7     PreFilter
            // bit 8       Span gas check enable
            // bit 9       DacMOn enable
            // bit 10-11   Clamp, set bit to clamp reading to 00 = 200%, 01 = 150%, 10 = 125%, 11 = 100%
            // bit 12      DET1 reading format, 0 = %v/v 1 = %fsd
            // bit 13      DET2 reading format, 0 = %v/v 1 = %fsd
            // bit 14      liveData reading type, 0 = float, 1 = int
            // bit 15      Single channel emulation of calibration

   // 3 ranges
   double SensorFsd[3];                         // CH4L, CH4H, Propane
   double ZeroOffset;                           //
   double ZeroCalTemperature;                   //
   double SpanCalTemperature[3];                // CH4L, CH4H, Propane
   double PosZeroSuppress;                      //
   double NegZeroSuppress;                      //
   double CalGas[3];                            // CH4L, CH4H, Propane
   double SpanOffset[3];                        // CH4L, CH4H, Propane
   double El[3];                                // CH4L, CH4H, Propane
   double Power[3];                             // CH4L, CH4H, Propane
   double Rounding[3];                          // CH4L, CH4H, Propane
   double Filter0;                              // Most filtering
   double Filter1;                              //
   double Filter2;                              //
   double Filter3;                              //
   double Filter4;                              // Least filtering
   double FilterChangeHighTemp;                 // Above 60 degC
   double FilterChangeGas;                      //
   double FilterChange;                         //
   unsigned int  BaudRate;                      // comms speed
   unsigned int  WarmUpTime;                    // seconds
   double TemperatureOffset;                    // analogue temperature IC
   double CalPressure;
   double PressureCoefficientsM[4];
   double PressureCoefficientsC[4];
} ConfigData;
```

Description of Configuration parameters

Configuration Parameter	Description	Default	Remark
Version	A number used to identify the structure		
SensorType	Text giving details of gas type.		Maximum of 8 characters
ModeBits	Option used to select gas ranges in use. Note: only applicable for certain sensors bit 0 - voltage out '0', bridge out '1' bit 1 - range 1 enable '1'	Sensor enabled	Should not be changed by the user. The analogue output type is set at the build stage and cannot be changed at a later date
SensorFsd	The upper limit for the gas range expressed in %v/v or ppm terms.	Based on sensor type	Must be an integer value
ZeroOffset	Value that is applied to adjust the sensor to read zero when no gas is present. This is an offset from the initial calibration factor during sensor testing.		Set during the zero command
ZeroCalTemperature	The sensor temperature when the last zero was performed		Set during the sensor zero command
SpanCalTemperature	The sensor temperature when the last span was performed		Set during the sensor span command
DacZero	The analogue output for zero gas		Expressed in DAC counts, 0-0xFFFF
DacFsd	The analogue output for gas at FSD		A calibration factor to convert gas readings to an analogue output representing the maximum gas level that can be detected accurately by the sensor
PosZeroSuppress	Readings between zero and this level will be set to zero.	0	
NegZeroSuppress	Readings between zero and this level will be set to zero.	0	
CalibrationGasValue	The gas level used for the last calibration		Set during the sensor span
SpanOffset	Value that is applied to adjust the sensor to read gas correctly. This is an offset from the initial calibration factor during sensor testing		Set during the sensor span
EI (Linearity)	Constant used to convert sensor output to a linear value.		An average value that is determined during initial testing of the sensor, based on the physical construction of the sensor
Power	Constant used to convert sensor output to a linear value.		An average value that is determined during initial testing of the sensor, based on the physical construction of the sensor
SerialNumber	A unique serial number		Assigned during manufacture
Rounding	The sensor resolution		Changing this value will not increase the sensor resolution. It is used to smooth out fluctuation in the gas readings
DacPowerup	The analogue output for the sensor warm-up time		This should be at a level that is less than the zero gas level which will allow a faulty sensor to be detected.
BaudRate	Serial data transfer speed	38400	Valid settings are 4800, 9600, 19200 and 38400. Bits 0 and 1 select the baud rate – other bits should be left intact
WarmUpTime	Number of seconds before the sensor measures gas, however some sensors take up to 60 seconds before fully stable. The analogue output is held at the start up voltage during this time.	15	Sensors can take up to 60 seconds t stabalise. A value that is too low will result in spurious readings during the first 60 seconds.

2.6 Live data structure

The live data takes the following forms:

```
Struct {
        unsigned int Version;                   // 1
        unsigned int StatusFlags;
        double Reading;
        double Temperature;
        unsigned int Det;
        unsigned int Ref;
        double Fa;
} LiveData;                                     // 18 bytes

struct {
        unsigned int Version;                   // 1
        unsigned int StatusFlags;
        double Reading;
        double Temperature;
        unsigned int Det;
        unsigned int Ref;
        double Fa;
        long Uptime
} LiveData;                                     //  24 bytes

struct {
        unsigned int Version;                   // 1
        unsigned int StatusFlags;
        double Reading;
        double Temperature;
        unsigned int Det;
        unsigned int Ref;
        double Fa;
        long Uptime;
        unsigned int DetMin;
        unsigned int DetMax;
        unsigned int RefMin;
        unsigned int RefMax;
} LiveData;                                     // 32 bytes

struct {
        unsigned int Version;                   // 3
        unsigned int StatusFlags;
        double Reading1;
        double Temperature;
        double Reading2;
        double Det1;
        double Ref;
        double Fa1;
        long Uptime;
        double Det2;
        double Fa2;
        unsigned int StatusFlags2;
        double Reading3;
} LiveData
```

```
struct {
        unsigned int Version;                   // 4
        unsigned int StatusFlags;
        double ReadingFloat;
        double Temperature;
        unsigned int Det1;
        unsigned int Ref;
        double Fa;
        long Uptime;
        uusigned DetMin;
        uusigned DetMax;
        uusigned RefMin;
        unsigned int RefMax;
} LiveData

struct {
         unsigned int Version;                  // 5
         unsigned int StatusFlags;
         signed int Reading;
         unsigned int ReadingMultiplier;
         double Temperature;
         unsigned int Det;
         unsigned int Ref;
         double Fa;
         long Uptime;
         unsigned int DetMin;
         unsigned int DetMax;
         unsigned int RefMin;
         unsigned int RefMax;
} LiveData;
```

Version 5 Livedata has the 4-byte float reading replaced by 2 integers to allow the user to obtain the gas reading by applying simple division rather than converting using the IEEE-754 standard

The gas readings are scaled according to FSD as follows:

FSD	Multiplier
>5000	1
>2000	2
>1000	8
>100	16
>60	128
>20	256
>10	512
>5	1024
>2	2048
else	4096

Thus the gas reading field must be divided by it's multiplier to obtain the actual reading.

Example 1:
Gas reading bytes 0xEB11 = 0x11EB = 4587
Multiplier bytes 0x0008 = 0x0800 = 2048

Calculated gas = 4585/2048 = 2.24%

Example 2:
Gas reading bytes 0xAFFF= 0xFFAF = -81
Multiplier bytes 0x0004 = 0x0400 = 1024

Calculated gas = -81/1024 = -0.079%

Provisional structure – under development

```
struct {
        unsigned int  Version;                          // Version 6

        unsigned int  StatusFlags;
         union
         {
                double ReadingFloat;
                unsigned long ReadingInt_Mult;          // int reading + int multiplier occupies 4 bytes
         }Reading1;
         Double Temperature;
         union
         {
                double ReadingFloat;
                unsigned long ReadingInt_Mult;          // int reading + int multiplier occupies 4 bytes
         }Reading2;
         Double  Det1;
         Double  Ref;
         Double  Fa1;
         Unsigned long Uptime;
         Double  Det2;
         Double  Fa2;
         unsigned int  StatusFlags2;
         union
         {
                Double  ReadingFloat;
                unsigned long ReadingInt_Mult;          // int reading + int multiplier occupies 4 bytes
         }Reading3;
} LiveData2;


struct {
         unsigned int  Version;                         // Version 7
         unsigned int  StatusFlags;
         union
         {
                Double ReadingFloat;
                Unsigned long ReadingInt_Mult;          // int reading + int multiplier occupies 4 bytes
         }Reading1;
         Double  Temperature;
         union
         {
                Double ReadingFloat;
                Unsigned long ReadingInt_Mult;          // int reading + int multiplier occupies 4 bytes
         }Reading2;
         Double Det1;
         Double Ref;
         Double Fa1;
         Unsigned long Uptime;
         Double Det2;
         Double Fa2;
         Unsigned int StatusFlags2;
         union
         {
                Double ReadingFloat;
                unsigned long ReadingInt_Mult;          // int reading + int multiplier occupies 4 bytes
         }Reading3;
         unsigned int StatusFlags3;
         unsigned int StatusFlags4;
} LiveData2;
```

Description of Live data

Configuration Parameter	Description	Default	Remark
Version	A number used to identify the structure		
StatusFlags	The current state of the sensor, usually zero for healthy operation	0	There may be multiple status flags, 0xFFFF expected value for internal faults
Reading	The current gas reading		There may be more than 1 reading
Temperature	The internal sensor temperature, usually 5 to 10 degC above ambient.		
Det	The sensor detector signal expressed in AtoD counts.		Reduces with gas
Ref	The sensor reference signal expressed in AtoD counts.		
Fa	The Fractional absorbance value which represents the gas level		Value from 0 to 1 representing gas
Uptime	The approximate time since the sensor was powered in 1/100 th's of a second		Maximum number is 4294967295 equivalent to approximately 497 days.
DetMin	The sensor detector minimum signal expressed in AtoD counts.		The sensor signals are sinusoidal. Usually used for diagnostic purposes
DetMax	The sensor detector maximum signal expressed in AtoD counts.		
RefMin	The sensor reference minimum signal expressed in AtoD counts.		
RefMax	The sensor reference maximum signal expressed in AtoD counts.		

Note: structure 1 is 20 bytes in length.
structure 2 is 24 bytes in length and has an extra variable.
structure 3 is 32 bytes in length and has 5 extra variables.

The Version number remains the same even when extra variables are added to the structure. Thus if software is written to accept structure 1 but receives more data, i.e. structure 2 or 3, then it simply ignores the extra bytes.

The StatusFlags field give the user additional information about the sensor as follows:

2.7 Status Flags

Live Data Version 1

Status Flags Name	**Bit**	**Details**
SIGNAL_TIMEOUT	**0x0001**	**Loss of pyro signal**
SIGNAL_NOISE	0x0004	Pyro signals changing too quickly
DET1_LOW	0x0040	Detector pyro signal below minimum setting
REF_LOW	0x0080	Reference pyro signal below minimum setting
VMON_ERROR	0x0800	Supply voltage outside limits, 3 to 5 volts
CONFIG_CSUM	0x1000	Configuration data corrupted
PRIVATE_CSUM	0x2000	Private data corrupted
USER_EEP_CSUM	0x4000	User data corrupted
PROG_CSUM_ERROR	0x8000	Firmware corrupted

Live Data Version 1, 4 & 5 V6 Firmware

Status Flags Name	**Bit**	**Details**
SIGNAL_NOISE	0x0004	Pyro signals changing too quickly
DET1_LOW	0x0040	Detector pyro signal below minimum setting
REF_LOW	0x0080	Reference pyro signal below minimum setting
VMON_ERROR	0x0800	Supply voltage outside limits, 3 to 5 volts
CONFIG_CSUM	0x1000	Configuration data corrupted
PRIVATE_CSUM	0x2000	Private data corrupted
WARM_UP	0x4000	Sensor readings should be ignored
PROG_CSUM_ERROR	0x8000	Firmware corrupted

The flags are added together to form one integer result, 2 bytes. If the sensor signals are too low then the returned flag word would be 0x00C0.

Status Flags 2 Name	**Bit**	**Details**
DET2_LOW	0x0010	Detector 2, dual sensor, pyro signal below minimum setting
WARM_UP	0x8000	Sensor readings should be ignored

The flags are added together to form one integer result, 2 bytes.

Live Data Version 7 V6 Firmware

Status Flags 2 Name	Bit	Details
DET2_FLT	0x0010	Detector 2, Dual sensor, pyro signal below minimum setting
FILTER_CSUM	0x0020	Filter data corrupted
WARM_UP	0x8000	Sensor readings should be ignored

Status Flags 3 Name	Bit	Details
Not in use returns 0x0000		

Status Flags 4 Name	Bit	Details
Not in use returns 0xFFFF		

Live Data Version 7 V7 Firmware

Status Flags 2 Name	Bit	Details
TEMP_COMP_DATA_ERROR	0x0004	Temperature compensation data corrupted
MIN_MAX_TEMP_CRC_ERROR	0x0008	Min / Max temperature data corrupted
DET2_FLT	0x0010	Detector 2, Dual sensor, pyro signal below minimum setting
FILTER_CRC	0x0020	Filter data corrupted
WARM_UP	0x0040	Sensor readings should be ignored
STATUS_4_ERROR	0x0080	Internal processor error
MAX_GAS_LIMIT_EXCEEDED	0x0100	Calculated gas readings outside sensor limits

Expected value = 0x0040, during warm-up period : 0x0000, after warm-up period

Status Flags 3 Name	Bit	Details
TEMP_COMP_DATA_CH4Z_CRC_ERROR	0x0001	Temperature compensation data corrupted
TEMP_COMP_DATA_CH4LS_CRC_ERROR	0x0002	Temperature compensation data corrupted
TEMP_COMP_DATA_CH4HS_CRC_ERROR	0x0004	Temperature compensation data corrupted
TEMP_COMP_DATA_HCS_CRC_ERROR	0x0008	Temperature compensation data corrupted
TEMP_COMP_DATA_CO2Z_CRC_ERROR	0x0010	Temperature compensation data corrupted
TEMP_COMP_DATA_CO2S_CRC_ERROR	0x0020	Temperature compensation data corrupted
TEMP_COMP_TEMP_CH4Z_CRC_ERROR	0x0040	Temperature compensation data corrupted
TEMP_COMP_TEMP_CH4LS_CRC_ERROR	0x0080	Temperature compensation data corrupted
TEMP_COMP_TEMP_CH4HS_CRC_ERROR	0x0100	Temperature compensation data corrupted
TEMP_COMP_TEMP_HCS_CRC_ERROR	0x0200	Temperature compensation data corrupted
TEMP_COMP_TEMP_CO2Z_CRC_ERROR	0x0400	Temperature compensation data corrupted
TEMP_COMP_TEMP_CO2S_CRC_ERROR	0x0800	Temperature compensation data corrupted
EEPROM_WRITE_TEST_ERROR	0x1000	Non volatile memory error

Expected value = 0x0000

Status Flags 4 Name	Bit	Details
RAM_TEST_ERROR	0x0001	Internal processor error
ADDRESS_TEST_ERROR	0x0002	Internal processor error
MATH_TEST_ERROR	0x0004	Internal processor error
STACK_TEST_ERROR	0x0008	Internal processor error
OSCILLATOR_TEST_ERROR	0x0010	Internal processor error

Expected value = 0xFFFF

2.8 User data structure

The sensor has 32 bytes of spare data that has been assigned to the user. It is up to the user to allocate this memory. This data has no effect on the sensor operation.

The data is accessed through command variable id 11, it can be both read and written.
In its simplest form the data sent and returned takes the following form:

```
struct{
        unsigned char   data[32];
}userData;
```

Note: The user can assign the data to any format, providing it takes no more than 32 bytes.

Note: The user data area is not available in the EN50271 type sensors

2.9 Gas level data structure

The data is passed with command variable id 3, is write only. This is to pass the calibration gas level.

Single gas sensor

```
struct{
        double          GasValue;
}calGas;
```

Dual gas sensor

```
struct{
        double          GasValue;
        unsigned int    Range;
}calGas;
```