Document Number		DTN0019	
Title		MoTeC Communications and Setup	
Approved By		*JA*	
Revision	**Date**	**Prepared By**	**Change History**
1.0	07/05/2009	DR	Initial

Contents

Introduction

This document describes how data communications works and how to setup communications to and from MoTeC devices.

Scope

This document applies to CAN and RS232 based communications, in particular in relation to MoTeC's Data Loggers: ADL, ADL2, ADL3 and ACL.

Communications Basics

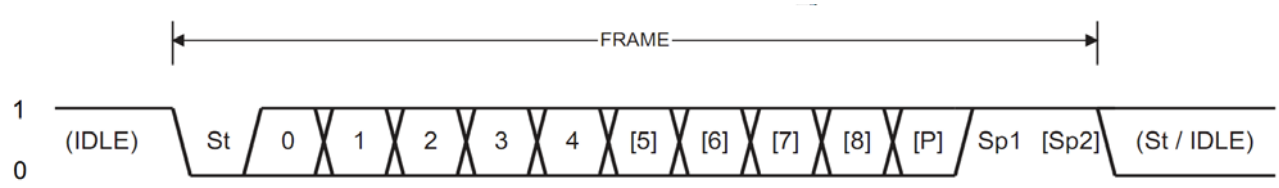
Data is sent from one electronic device to another by serial communication. There are various protocols all working on the principal of a voltage level representing a binary number of 0 or 1. This voltage level being present for an amount of time represents one bit of data. Each bit of data is sent sequentially one after another. The number of bits transferred per second is the Baud rate (bits per second - bps). Often extra time synchronising and checksum bits are added to increase data transfer quality and noise immunity.

RS232

This is a point-to-point type communication where one device transmits while the other device receives. Each device has separate transmit (tx) and receive (rx) pins and these must be connected from one device to the other for bi-directional communication. Both devices must be connected to a common ground. RS232 data is generally sent in a frame of 10 bits; a start bit (always '0'), 8 bits of data and a stop bit (always '1'). Frame lengths can vary by the number of data bits or stop bits.

After one frame (byte) has been sent, another is sent immediately or the line sits idle. When a stream of data (or 'packet') is sent, it starts with a 'header' identifying the beginning of the packet. The receiving device counts from the header to determine which bits and/or bytes represent which channel.

Typical RS232 Baud rates range from 1200 to 115200 bits per second (bps).



St; Start bit, always low.

(n); Data bits (0 to 8).

P; Parity bit. Can be odd, even or none.

Sp; Stop bit/s, always high.

IDLE; No transfers on the communication line.

[ ]; Optional bit.

Controller Area Network (CAN)

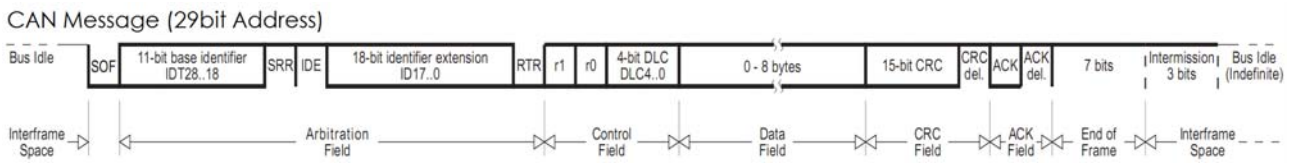
CAN is a bus type communication where many devices or 'nodes' can transmit and receive data to and from any other node on the bus. The bus normally consists of a twisted pair of wires; CAN High and CAN Low. Nodes are simply spliced in.

Both wires carry the same data. They are used as a twisted pair to shield each other from noise and to improve high speed transfer.

One CAN message can be over 130 bits long and consists of a CAN address, up to 64 data bits and several timing and checksum bits.

The CAN address is an identifier for the data sent. Each node is pre-programmed to either ignore or receive the message at a particular address and interpret the data. The address length is either 11 bits (standard) or 29 bits (extended).

All nodes can simultaneously receive data, but only one node can send data at a time. Normally, if a node wants to send data, it waits until the bus is idle and then begins transmission. However, if two or more nodes are waiting to send data they could start sending at the same time. To avoid a collision the node sending the lowest value address takes priority and continues sending. The other nodes sense this, give up sending and wait for the bus to be idle again.



Bits, Bytes, Binary and Hexadecimal

Electronic devices use binary numbers to communicate as it is easy to differentiate between a low and high voltage level; usually 0 and 5 volt represent 0 and 1.
When counting in binary, since each digit can only have two different values a binary number grows long in digits very quickly making them hard to read. They are often represented as hexadecimal (hex) numbers, using 16 different values per digit. One hex digit represents exactly 4 binary digits, so an 8 digit binary number can be represented with a 2 digit hex number that is much easier to read.

To differentiate between decimal, hex and binary numbers they are written with a preceding '0x' for hex and '0b' for binary, or noted with hex or bin somewhere in the area.

A bit is simply one binary digit that has a value of 0 or 1.

A byte is an 8 digit binary number that has a value of 0 to 255 (or 0x00 to 0xFF, or 0b00000000 to 0b11111111).

The table shows binary and hexadecimal numbers against decimal numbers.

Decimal	Hex	Binary
0	00	00000
1	01	00001
2	02	00010
3	03	00011
4	04	00100
5	05	00101
6	06	00110
7	07	00111
8	08	01000
9	09	01001
10	0A	01010
11	0B	01011
12	0C	01100
13	0D	01101
14	0E	01110
15	0F	01111
16	10	10000
17	11	10001
18	12	10010

Bit/Byte Numbering and Offset

Within a string of data each byte is given a number and so is each bit in the byte. MoTeC's numbering system starts at '0' for bits and bytes. The first data byte in a string is 'Offset 0'. In a CAN message this is the first byte of the data field, in a RS232 packet this is the first byte after the header. For Delimited format the data after the header but before the first delimiter is 'Variable 0' (see Data Formats).

Bytes are sent most significant bit first in CAN and least significant bit first in RS232.

CAN Bit #	76543210	76543210	76543210	76543210	76543210	76543210	76543210	76543210
Byte Offset #	0	1	2	3	4	5	6	7
Time of bit 7	0 µs	+8 µs	+16 µs	+24 µs	+32 µs	+40 µs	+48 µs	+56 µs

Example of one CAN data field. Bits are transmitted / received reading from left to right at 1 Mbps.

Channel Lengths, Signing and Decimal Places

Channels can have varying lengths of data bits, which is the amount of space they occupy in memory. Most channels in MoTeC loggers are 16 bits in length, but where high resolution or very large numbers are needed, they can be 32 bits long, e.g. GPS lateral and longitudinal, or Device Up Time.

16 bit channels have 65536 possible different values.

32 bit channels have 4294967296 possible different values.

Only positive integers can exist in a data stream. Representation of fractional numbers is done by placing a decimal point in the appropriate place at the user's end. Negative numbers are represented by calling the channel 'Signed'. For the device, signing and decimal points do not exist; all data is treated as integers, no decimal points. The exception being delimited decimal format where the ASCII characters '-' (minus sign) and '.' (dot/period) are read under certain conditions.

A signed 16 bit channel has a value between -32768 and 32767. If the channel has for example 2 decimal places, the value will be between -327.68 and 327.67.

The most significant bit in a signed channel is the sign, with a value of '0' for '+' (positive) and '1' for '-' (negative).

An Unsigned 16 bit channel has a value between 0 and 65535 (0x0 and 0xFFFF).

Values between 0 and 32767 (or 0x0 and 0x7FFF) are equal for Signed and Unsigned channels, but for values between 32768 and 65535 (or 0x8000 and 0xFFFF) the Signed channel equals the value of the Unsigned channel minus 65536 (0x10000).

Note: All channels in MoTeC loggers are Signed.

Unsigned	0xFFFC	0xFFFD	0xFFFE	0xFFFF	0x0000	0x0001	0x0002	0x0003	0x0004
Signed	-4	-3	-2	-1	0	1	2	3	4

Bit Masking

Bit Masking is an operation where a logical ‘AND’ is performed between two binary numbers.

For every bit in the Mask with value ‘1’, the corresponding bit of the Received Data is passed to the result. A Mask with all bits set to '1' lets the Received Data pass unchanged and can be considered as No Mask. For every bit in the Mask with value '0', the corresponding bit in the Received Data is passed as '0'.

This is useful if multiple statuses are combined in one channel. If for example the second bit in the received Data contains the status that needs to be read, the mask is set to all zero except for the second bit.

Bit Masking can also be used for channels that are not whole bytes long, e.g. 4 bits (nibble), 2 bits (crumb) or 10 bits. The remaining bits of the byte, which do not belong to the channel being received, can be ignored or ‘Masked off’.

Bit Masks are defined in hexadecimal, but are here also shown in binary.

	Nibble	**Crumb**	**Bit / Status**	**No Mask**	**10 bit Channel**
Bit Mask (hex)	0F	0C	40	FF	03 FF
Bit Mask (bin)	00001111	00001100	01000000	11111111	00000011 11111111
Received	0110**1101**	1011**01**10	0**1**011011	**10111001**	00101**01 11010001**
Result	00001101	00000100	01000000	10111001	00000001 11010001
Channel	13	8	64	185	465

In this example for Status, the result comes through in the original bit position. This results in a channel value of ‘0’ when the status is off and ‘64’ when the status is on. To give the correct status value of 0 or 1 the channel divisor needs to be set to 64.

Similarly, in this example for Crumb the possible values are ‘0’, ‘8’, ‘16’ or ‘24’ and need to be divided by ‘8’ to give the correct channel value of ‘0’, '1', '2' or ‘3’.

Scaling

Data transmitted has no unit or resolution transmitted with it, it is just a number. To turn this number into something usable it needs a unit of measurement and a resolution added after it is received. Each channel has a base unit and resolution. If the unit or resolution of a channel transmitted differs from this base value, the data needs to be scaled accordingly. By setting a Multiplier, Divisor and Adder, the calculations are applied to the received data (in respective order) before loading the value into the channel. All calculations are performed as if there were no decimal places. For example a channel value of 3.6 mm is transmitted as the number ‘36’. If the receiving channel resolution is 0.001 mm then ‘36’ would be multiplied by ‘100’ to equal ‘3600’ to fit the 3 decimal place channel 3.600 mm.

Channels can also be scaled before transmitting using the exact same method.

	Channel sent	Data	Multiplier	Divisor	Adder	Channel received
Engine RPM	3379 rpm	0x0D33	1	6	0	56.3 Hz
Engine Temp	180 °F	0x00B4	5554	1000	-177	82.2 °C
Battery Volts	13.4 V	0x0086	10	1	0	13.40 V
Oil Pressure	58.24 psi	0x16C0	1000	1450	0	401.6 kPa
Air Temp *(scaled)	34 °C	0x6C	10	2	-200	34.0 °C

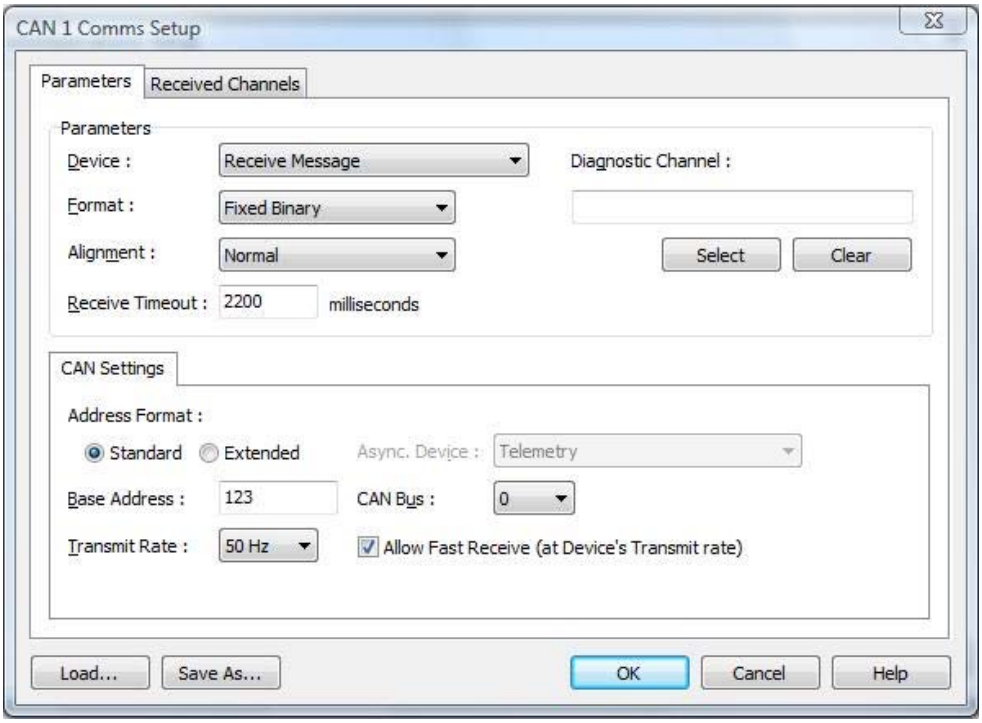
** Air Temp has special scaling: 1 bit = 0.5 °C, offset 20 °C, range -20 to 105 °C, 1 byte long.*

Channels must always be scaled to the base units. If a different unit is required, the conversion is done in the background when the unit is selected in Edit Channels.

Only Integers can be entered for scaling with values from -32768 to 32767.

Decimal Format also has a 'Modulus' setting. After all other operations, the value is divided by the modulus and the result is the remainder. '0' is no Modulus.

Advanced Comms Setup



Parameters

Device: The Device setting tells the logger what message header data or special format to expect.

The following generic types exist in CAN:

Receive Message: Receives data on the CAN address specified.

Transmit Message: Transmits channels on the CAN address specified.

Receive Message Block: Receives data in a block of 16 consecutive CAN addresses. See details under *Message Types*

Async Expander: Receives an RS232 type data stream on CAN. The RS232 type needs to be selected as well.

Format:

Data can be represented as binary or ASCII. Binary is the numeric value of the byte. With ASCII the numeric value represents a character (see Appendix 2 for the ASCII character set). This is method allows for sending more than just numbers. MoTeC devices only read numbers and relevant symbols.

The Formats supported by MoTeC Data Loggers are:

Fixed Binary: The binary value of the byte is the channel value.

Byte Offset	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Data (Hex)	0x53	0x25	0xB5	0x2F	0x45	0xFF	0x7A	0x03
Channel Value	83	37	181	47	69	255	122	3

Example shows fixed binary, channel length of '1' byte.

Fixed Hex: The ASCII value of the byte as hex (0 to 9 and A to F ASCII) is the channel value.

Fixed Decimal: The ASCII value of the byte as decimal (0 to 9 ASCII) is the channel value

Byte Offset	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Data (Hex)	0x31	0x41	0x32	0x30	0x46	0x43	0x37	0x35
Data (ASCII)	1	A	2	0	F	C	7	5
Channel Value	1	10	2	0	15	12	7	5

Example shows Fixed Hex with channel length of '1' byte. Fixed Decimal is very similar.

Delimited Hex: The 'Variable' is the number of delimiters after the 'header' data. Delimiters are ASCII ','(comma) or ';'(semicolon). The channel value is the ASCII string as hex (0 to 9 and A to F) between delimiters.

Delimited Decimal: The 'Variable' is the number of delimiters after the 'header' data. Delimiters are ASCII ','(comma), ';'(semicolon) or '*' (asterisk) . The channel value is the ASCII string as decimal (0 to 9) between delimiters. Decimal point ('.') and minus ('-') characters are also read in certain circumstances.

Variable #	**0**	**1**	**2**	**3**
Data (Hex)	0x31 0x2C	0x39 0x31 0x2C	0x34 0x31 0x2C	0x34 0x30 0x33 0x2C
Data (ASCII)	1,	91,	41,	403,
Channel Value	1	91	41	403

Example shows delimited decimal, ASCII number followed by ',' (comma) delimiter.

Fast Binary (16 bit Normal): The same as Fixed Binary except alignment is fixed as 'Normal' and channel length is fixed at '2' (16 bit). This uses less CPU time.

Note: With ASCII formats, any data that is not a number or specific symbol is ignored except for delimited decimal format. If in delimited decimal format the first byte after the delimiter is invalid, all characters binary values are read and added together until the following delimiter.

Alignment:

For channels that are longer than 8 bits, the binary number is broken into bytes. The Alignment setting determines if the most significant byte or least significant byte is transmitted or received first.

Note: Only valid for Fixed Binary and Fixed Hex.

Normal Alignment (aka Motorola or Big Endian); The **Most** significant byte is sent/received first.

Word Swap Alignment (aka Intel or Little Endian); The **Least** significant byte is sent/received first.

		Normal		Word Swap	
	16 bit Data Channel	Offset 0	Offset 1	Offset 0	Offset 1
Binary	0b1110000100001011	0b11100001	0b00001011	0b00001011	0b11100001
Hex	0xE10B	0xE1	0x0B	0x0B	0xE1

Receive Timeout:

The receive timeout is how long the device waits for the next packet of data before changing the channel to its default setting and flagging a no data error (512) in the comms diagnostic channel. Time is set in 1 msec increments up to 30 seconds. The default timeout is 2200 msec.

Timeout should be set to at least 2 times the transmitted rate, e.g. Data sent at 10 Hz rate is transmitted every 100 msec. The Timeout should be set to at least 200 msec.

Some devices send data very infrequently, e.g. once every 5 seconds requiring a Timeout of at least 10 seconds.

RS232 Settings

All settings must match the other connected device for communications to succeed.

Baud Rate: The speed at which the data is transferred.

Data Bits: Number of data bits per frame, normally 8

Stop Bits: Number of stop bits, normally 1.

Parity: Bit checksum, normally None.

Transmission Control Channel: Only transmits telemetry data when the selected channel is True.

Telemetry Settings

Most RS232 Device Types have the option to transmit telemetry as well as receive from another device, or send telemetry only.

Limit to Modem Carrier Rate; transmits data with enough idle time as not to overflow the bandwidth of a slower 'carrier' baud rate.

Streaming; normally telemetry is sent at a rate rounded to a whole number of samples per second leaving some idle time on the line after each packet. If the Streaming box is checked the data is forced to be sent without gaps, using 100% bandwidth.

CAN Settings

Address Format: Standard format is a 11 bit address, with values of 0x1 to 0x7FF. Extended format is a 29 bit address, with values of 0x1 to 0x1FFFFFFF.

Base Address: The CAN address on which the data is being transmitted or received.

Transmit Rate: The number of times per second a message is transmitted (ignored for receive).

CAN Bus: The physical bus on which the message is transmitted (ACL and ADL3 Data loggers only).

Allow Fast Receive: The channels are updated at the same rate as the message is received, allowing for log rates up to 1000 Hz.

Async Device: Receive RS232 type data format on CAN (Async Expander device type only).

Message Type

Single: With single message type, the position of each byte in the data string relates to the channel assigned.

Compound: With compound message type there is a message identifier within the data section of the message or packet. The value of the ID relates the rest of the data to a unique set of channels. This

message type allows for communicating a large number of channels on a single CAN address. When transmitting a compound message the Transmit Rate is the rate at which each message is transmitted, meaning each channel is transmitted at the Transmit Rate divided by the number of ID sets.

Compound Settings

Offset: The number of bytes after the header or CAN address where the most significant byte of the ID is located in the data.

ID: The value of the identifier in hex.

ID Mask: A bit mask for the ID. (See Bit Masking). Has no effect on transmit message.

Offset	0 1	2 3	4 5	6 7
Msg 0x123	ID = 0x0000	Chan 1	Chan 2	Chan 3
Msg 0x123	ID = 0x0100	Chan 4	Chan 5	Chan 6
Msg 0x123	ID = 0x0200	Chan 7	Chan 8	Chan 9
Msg 0x123	ID = 0x0300	Chan 10	Chan 11	Chan 12

Example of transmitting twelve 16 bit channels on one CAN Address
(This is the M800 ADL receive format).

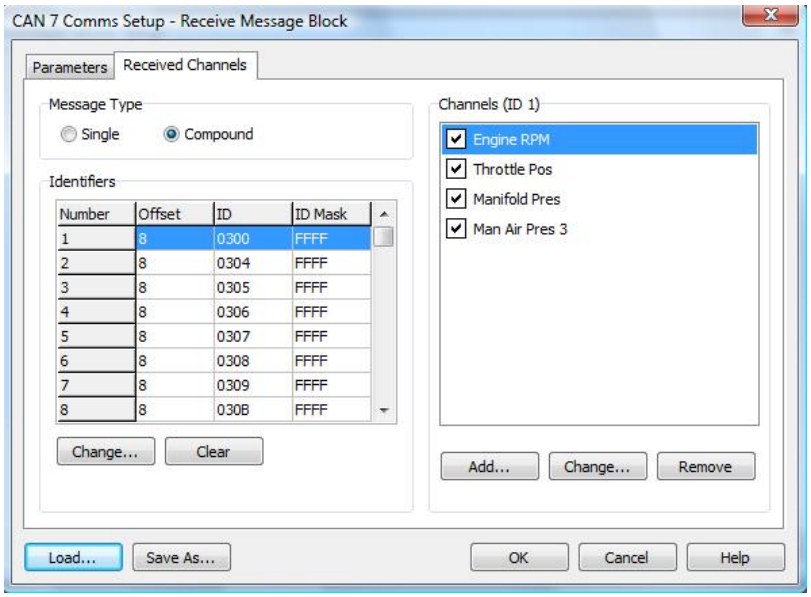
When transmitting a compound message the data is written to the message after the ID. Data can therefore be written over the ID. Care must be taken not to do this, which is as easy as not transmitting data on the same offset as the ID.

Often a two byte identifier is not needed. In this case, one of the two bytes can be overwritten with data. This will work as long as there is still at least one identifying marker in one byte with the receiving end configured to read only this byte.

Alignment settings do not affect the identifier; it is always transmitted as Normal - most significant byte first.

Receive Message Block Device Type Setup

Receive Message Block uses the compound ID setup to define which message of the block relate to which set of channels. The Identifiers Offset must all be set to '8', and Identifiers Bit Mask all set to 'FFFF'. The Identifiers ID contains the CAN Address of each message to be received in the block, and must be in the range of 'Base Address' to 'Base Address + 0xF'. For each Compound ID, channels are selected the same as if it were a Single Message.



Receive Message Block, base address 0x300

Channel Settings

Default Value: Channel value when comms is in error (Diag is not '0')

Offset: The Byte number in the data stream the first byte of the channel occurs.
CAN messages - '0' to '7' are valid.
Single type RS232 packet - '0' to '255' is valid.
Compound Type RS232 packet - '0' to '15' is valid.

Variable: Delimited format only. Specifies how many delimiters through the packet the channel occurs.
Delimited Hex - '0' to '30' is valid.
Delimited Decimal - '0' to '70' is valid.

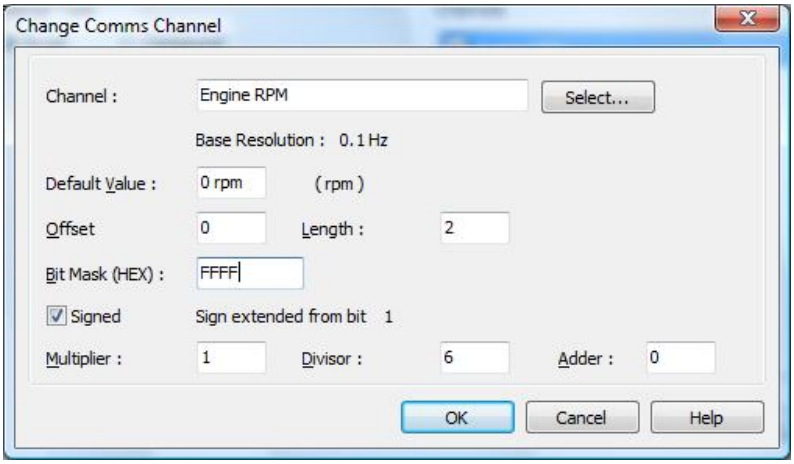
Length: The number of bytes long to be received as one channel.
Fixed binary format '1', '2' or '4' bytes are valid. ('4' not valid for ADL2)
Fixed Hex/Decimal format '1' to '16' bytes is valid.
Delimited format length is between delimiters, not applicable.

Decimal Format: For Delimited Decimal format only. Specifies the resolution of the received data by reading the decimal point character in the ASCII string. Used if trailing zeros are omitted from the string. 'As Received' ignores the decimal point from the ASCII string.

Bit Mask: Binary Bits to logically 'AND' with the received channel. See Bit Masking section.

Signed Checkbox: Defines whether the channel is Signed or Unsigned. See *Channel Lengths, Signing and Decimal Places* Section.

Multiplier, Divisor and Adder: After the data has been received and bit masked, it is multiplied, divided and added in this order. This operation allows the data to be scaled if sent in a different unit and resolution to the base units and resolution of the assigned channel. See *Scaling*



Receive Message Fixed Binary, Engine RPM from M800 ECU. Note Base resolution 0.1 Hz, requires divide by 6 to convert from rpm 0dp to Hz 1dp

M800 ECU Comms Setup

The hundred series ECUs offer a low level of customisable communication. Predefined data sets and two custom data sets can be transmitted on CAN and RS232. Only predefined CAN templates can be received, which includes templates for all CAN-based MoTeC products, and NMEA GPS information on RS232.

Custom Data Sets

There are two custom data sets where a list of channels to be transmitted can be selected. Up to 64 channels can be selected per set. The 'Export Comms template' feature creates CAN templates of the custom data set that appear in the templates list of the selected devices. These templates require the custom data set to be transmitted from the M800 in the 'CRC32' format on a CAN Address of '1520'.

CAN Setup

Within the CAN setup parameters there are 7 sections, CAN 0 to CAN 6. Different data can be received and transmitted for these sections in a fixed format.

Custom data sets can be transmitted in three different fixed formats: Compound, Sequential, or CRC32.

Compound format transmits each group of 3 channels in custom data set item number order with an identifier at the start of each message. Transmitted on the CAN address specified until all channels are sent before repeating at the transmit rate defined.

Offset	0	1	2	3	4	5	6	7
CAN Address	0x0000		Item 1		Item 2		Item 3	
CAN Address	0x0100		Item 4		Item 5		Item 6	
CAN Address	0x0200		Item 7		Item 8		Item 9	

Example of transmitting 9 channels in compound format

Sequential format transmits each group of 4 channels in custom data set item number order on a sequentially incremented CAN address starting at the address specified. Up to 16 CAN addresses can be used for 64 channels, transmitted at the rate specified.

Offset	0	1	2	3	4	5	6	7
CAN Address	Item 1		Item 2		Item 3		Item 4	
CAN Address + 1	Item 5		Item 6		Item 7		Item 8	
CAN Address + 2	Item 9		Item 10		Item 11		Item 12	

Example of transmitting 12 channels in sequential format

CRC32 format transmits all channels in the MoTeC CRC32 format. This is an RS232 type format, beginning with a header and ending with a CRC32 32bit checksum. Data packet is sent on the CAN address specified. This format is used when transmitting custom data sets on telemetry (RS232).

Offset	0	1	2	3	4	5	6	7
CAN Address	0x82	0x81	0x80	Length	Item 1		Item 2	
CAN Address	Item 3		Item 4		Item 5		Item 6	
CAN Address	Item 7		Item 8		CRC 3	CRC 2	CRC 1	CRC 0

Example of transmitting 8 channels in CRC32 format

Deciphering Comms Setup Information

There is no consistency in terminology and numbering regarding Comms setup information. Differences can be many and varying. Sometimes trial and error is needed if information is limited and some common sense is required to decode the information to write a MoTeC template.

For example:

Byte and/or bit numbering can start at '1' instead of ‘0’ or the offset is in bits only, not bytes.

Scaling information can be presented in different ways: '0.125 units per bit' and '800 counts equals 100 units' represent the same scaling.

CAN Address can be called an ID. Compound ID can be called a Multiplexor.

The list goes on.

Appendix 1 - Diagnostic Channels and Error Counters

Diagnostic channels are used to display any error reported by the communications hardware.

The error codes are as follows:

0 No comm errors, receiving or transmitting data correctly.

RS232 hardware/comms errors

1 Parity is wrong

2 Framing error, no stop bit detected. Wrong baud rate, wrong number of bits or noise.

4 Noise, a glitch in data, probably caused by noise

8 Overrun, hardware issue.

General comms protocol errors

256 Bad Config, Invalid comms setting.

512 No data received before the 'Timeout' expired.

1024 Checksum error, comms protocol error.

2048 Wrong data, Check wiring.

CAN Bus Errors

4096 Bus Warning. Error counter is over 96 errors but bus may still function. Check wiring, terminating resistors and baud rate.

8192 Bus Off. Error counter is over 255, node switched off. Check wiring, terminating resistors and baud rate.

16384 CAN transmit warning. Check wiring, terminating resistors and baud rate.

CAN Error Counter channels (ACL V1.4 and ADL3 only) are linked to the CAN hardware, counting up for every incorrectly received or transmitted message, and counting down for every correctly received or transmitted message. The range is 0 to 255.

Appendix 2 - ASCII Standard Character set

Char	Dec	Hex	Description
NUL	0	0	Null character
SOH	1	1	Start of heading
STX	2	2	Start of text
ETX	3	3	End of text
EOT	4	4	End of transmission
ENQ	5	5	Enquiry, goes with ACK
ACK	6	6	Acknowledge
BEL	7	7	Bell
BS	8	8	Backspace
HT	9	9	Horizontal tab
LF	10	A	Line Feed
VT	11	B	Vertical tab
FF	12	C	Form Feed
CR	13	D	Carriage Return
SO	14	E	Shift Out
SI	15	F	Shift In
DLE	16	10	Data link escape
DC1	17	11	XON, with XOFF to pause listings
DC2	18	12	Device control 2
DC3	19	13	XOFF, with XON is TERM
DC4	20	14	Device control 4
NAK	21	15	Negative acknowledge
SYN	22	16	Synchronous idle
ETB	23	17	End transmission block
CAN	24	17	Cancel line
EM	25	19	End of medium
SUB	26	1A	Substitute
ESC	27	1B	Escape
FS	28	1C	File separator
GS	29	1D	Group separator
RS	30	1E	Record separator
US	31	1F	Unit separator
SP	32	20	Space
!	33	21	Exclamation mark
"	34	22	Quotation mark
#	35	23	Cross hatch (number sign)
$	36	24	Dollar sign
%	37	25	Percent sign
&	38	26	Ampersand
`	39	27	Closing single quote (apostrophe)
(	40	28	Opening parentheses
)	41	29	Closing parentheses
*****	42	2A	Asterisk (star, multiply)
+	43	2B	Plus
,	44	2C	Comma
-	45	2D	Hyphen, dash, minus
.	46	2E	Period
/	47	2F	Slant (forward slash, divide)
0	48	30	Zero
1	49	31	One
2	50	32	Two
3	51	33	Three
4	52	34	Four
5	53	35	Five
6	54	36	Six
7	55	37	Seven
8	56	38	Eight
9	57	39	Nine
:	58	3A	Colon
;	59	3B	Semicolon
<	60	3C	Less than sign
=	61	3D	Equals sign
>	62	3E	Greater than sign
?	63	3F	Question mark

Char	Dec	Hex	Description
@	64	40	At-sign
A	65	41	Uppercase A
B	66	42	Uppercase B
C	67	43	Uppercase C
D	68	44	Uppercase D
E	69	45	Uppercase E
F	70	46	Uppercase F
G	71	47	Uppercase G
H	72	48	Uppercase H
I	73	49	Uppercase I
J	74	4A	Uppercase J
K	75	4B	Uppercase K
L	76	4C	Uppercase L
M	77	4D	Uppercase M
N	78	4E	Uppercase N
O	79	4F	Uppercase O
P	80	50	Uppercase P
Q	81	51	Uppercase Q
R	82	52	Uppercase R
S	83	53	Uppercase S
T	84	54	Uppercase T
U	85	55	Uppercase U
V	86	56	Uppercase V
W	87	57	Uppercase W
X	88	58	Uppercase X
Y	89	59	Uppercase Y
Z	90	5A	Uppercase Z
[	91	5B	Opening square bracket
****	92	5C	Reverse slant (Backslash)
]	93	5D	Closing square bracket
^	94	5E	Caret (Circumflex)
_	95	5F	Underscore
`	96	60	Opening single quote
a	97	61	Lowercase a
b	98	62	Lowercase b
c	99	63	Lowercase c
d	100	64	Lowercase d
e	101	65	Lowercase e
f	102	66	Lowercase f
g	103	67	Lowercase g
h	104	68	Lowercase h
i	105	69	Lowercase i
j	106	6A	Lowercase j
k	107	6b	Lowercase k
l	108	6C	Lowercase l
m	109	6D	Lowercase m
n	110	6E	Lowercase n
o	111	6F	Lowercase o
p	112	70	Lowercase p
q	113	71	Lowercase q
r	114	72	Lowercase r
s	115	73	Lowercase s
t	116	74	Lowercase t
u	117	75	Lowercase u
v	118	76	Lowercase v
w	119	77	Lowercase w
x	120	78	Lowercase x
y	121	79	Lowercase y
z	122	7A	Lowercase z
{	123	7B	Opening curly brace
|	124	7C	Vertical line
}	125	7D	Closing curly brace
~	126	7E	Tilde (approximate)
DEL	127	7F	Delete (rubout), cross-hatch box