	Technical Note	
Document Number	TNAU0031	
Title	ADL Advanced Communications Setup	
Revision Date	**Prepared By**	**Approved By**
Draft 21/01/03	Andrew Naumann	Keith Turner

Introduction

The MoTeC ADL is capable of receiving data from another device – such as an ECU – in both RS232 and CAN formats. Motec has written a number of serial and CAN templates to simplify the setup for receiving data from another device by the ADL. The devices available can be seen by choosing the 'Select' option from the 'Inputs - Communications' screen under either RS232 or CAN1 – 6.

In other cases it may be necessary to change the communications settings, configure a CAN device etc. This document describes the parameters used for configuring the ADL that are found under the 'Advanced' setup screen for communications settings. While there are differences between RS232 and CAN, the use and definitions of the parameters are the same.

Note that it is usually not possible to define a template for an RS232 datastream that is not on the list. Whilst the format of CAN data is generally well defined, the coding of RS232 data is device dependant and a new template must be developed on a device by device basis by Motec. This can be a fairly involved process and costs may be incurred by the customer requesting a new or custom RS232 template.

Information on using a RTC to transmit additional serial data to an ADL by converting it to CAN data can be found in document "TNAU0035 RTC Serial Port".

Overview

There are a number of controllers and measuring devices that transmit serial data, often as RS232, but also as CAN data. The ADL has a built in serial port on which it can receive RS232 data.[1] The ADL can receive (or transmit) on up to 6 CAN base addresses. This can be from one or more other devices depending on how many base addresses each device uses for transmission – or if the ADL is transmitting to another device itself.

This document will focus on the use of the ADL for receiving CAN data. The Advanced comms screen is divided into 2 sections, 'Parameters' for the overall settings, and 'Received Channels'.

Contents:

[1] An external RTC with its own serial port can be used to add another serial data device by converting RS232 to CAN data.

Parameters

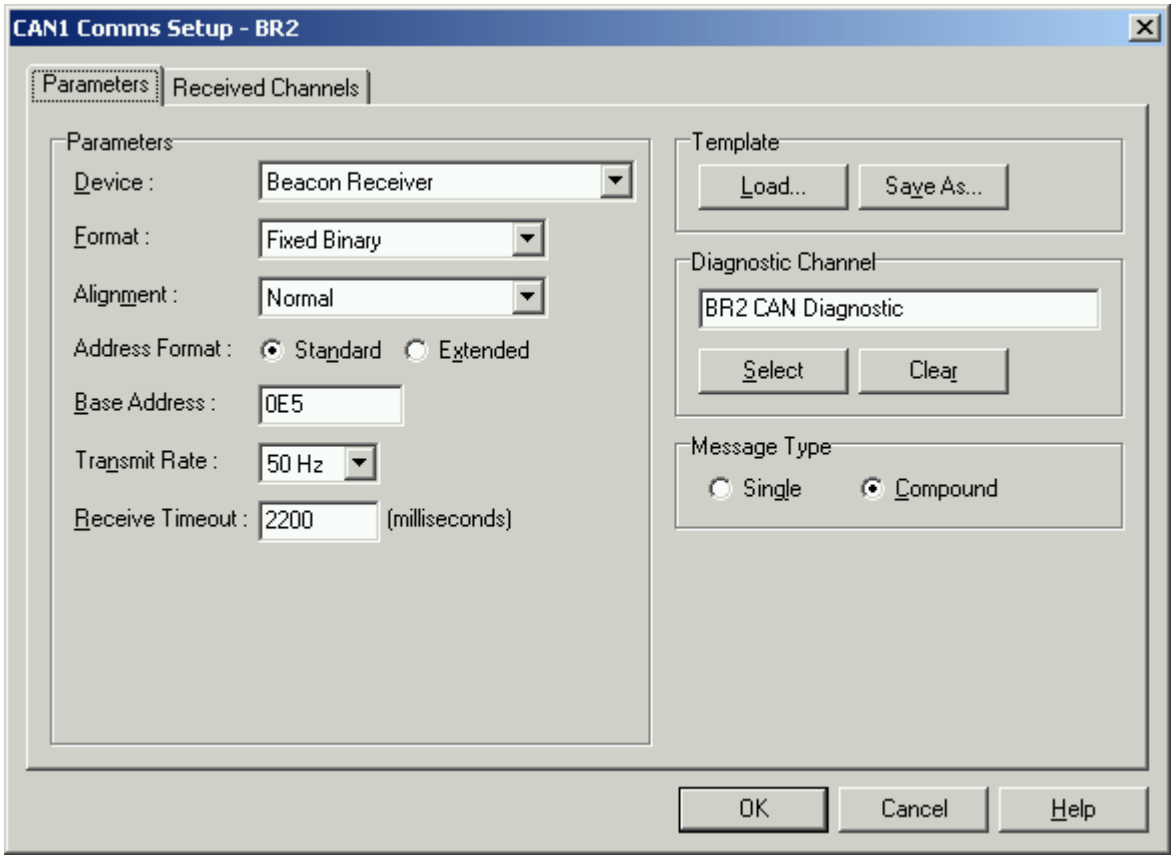

Fig 1: Advanced CAN comms parameters with the BR2 template selected

Device:
This tells the ADL what the format of the incoming data is. Ie: information about the data packet format. Different devices have different headers, checksums and other characters of varying lengths to describe and identify the data stream. For a 'new' CAN template a 'Received Message' device can be selected when reading data into the ADL.

Format:
The most common format for CAN data is fixed binary with the length of each channel specified under 'Received Channels'. This tells us that each packet will be of a fixed number of bytes (not specified here) with no separators between them. CAN data packets are fixed at 8 bytes long.

Fixed hex and fixed decimal are for non comma separated (non delimited) packets of fixed length that use ASCII characters. Ie: decimal being 0 – 9 with leading -/+ ignored and hex being 0 – 9, A – F.

Alignment:
This determines the byte order of received channels as all dash channels are treated as 16 bit values (2 bytes). Normal alignment is 'big endian' meaning that the most significant byte is received first, then the least significant.

Word Swap alignment is 'little endian' meaning that the least significant byte is received first, followed by the most significant.

Address Format:
Standard - 11 bit
Extended - 29 bit address

The address format should be specified in any protocol description, with most using Standard addressing. If base addresses used are less than 0x7FF[2] then this would indicate they are Standard length . See 'Base Address' below.

Base Address:
Also known as the Packet Identifier (ID), this is the 'address' associated with each data packet. Standard addressing (aka CAN 2.0A) uses 11 bits (range 0x000 – 0x7FF), while extended addressing (aka CAN 2.0B) uses 29 bits.

Transmit Rate:
If the ADL is transmitting data, this parameter sets the frequency at which this particular data packet is transmitted. Different rates may be assigned to different packets – ie: transmitted data with another base address. Higher priority packets can be transmitted more often reducing collisions on the data bus.

Receive Timeout:
Receive timeout is how long the dash will wait before populating a received channel with its default value.

For example, if the dash is configured to receive the engine temp channel via RS232 and this channel has a default value of 80 degrees, with receive timeout set to 1000 milliseconds. If no data is received on RS232 for 1000+ milliseconds, the engine temp channel will read 80 degrees (until data starts coming in on RS232 again).

Template
Comms configurations can be saved as files on a pc, copied and/or loaded into other configurations.

Diagnostic Channel
If there is a problem (eg: no data) then assigning a diagnostic channel can assist the user in determining the nature of the problem. A list of error codes can be found at the end of this document.

Message Type
Single or Compound – applies to both RS232 and CAN, though not often seen with RS232 data. Compound messaging is explained in more detail below. See also the document "PSAU0006 Compound Data Format".

Received Channels – Single Message

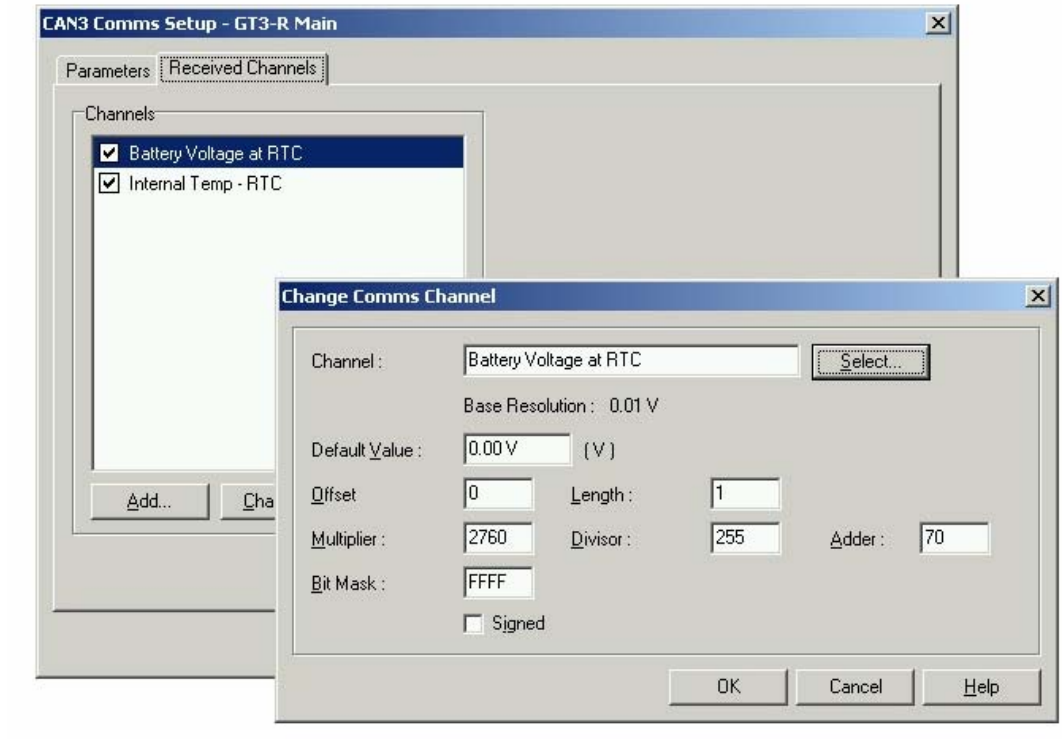

Fig 2: Single Message Type Received Channel Setup

[2] The notation '0x7FF' is used to indicate that the number after the '0x' is hexadecimal

Up to 8 bytes of information is transmitted in each CAN data packet. Received channels are 'Added' to the template by selecting them from the list of available channels. Required information for each channel is it's position (Offset) in the packet, Length in bytes and channel scaling , or 'units/bit'. See below for a description of the parameters in the Change Comms Channel dialog.

Received Channels – Compound Message

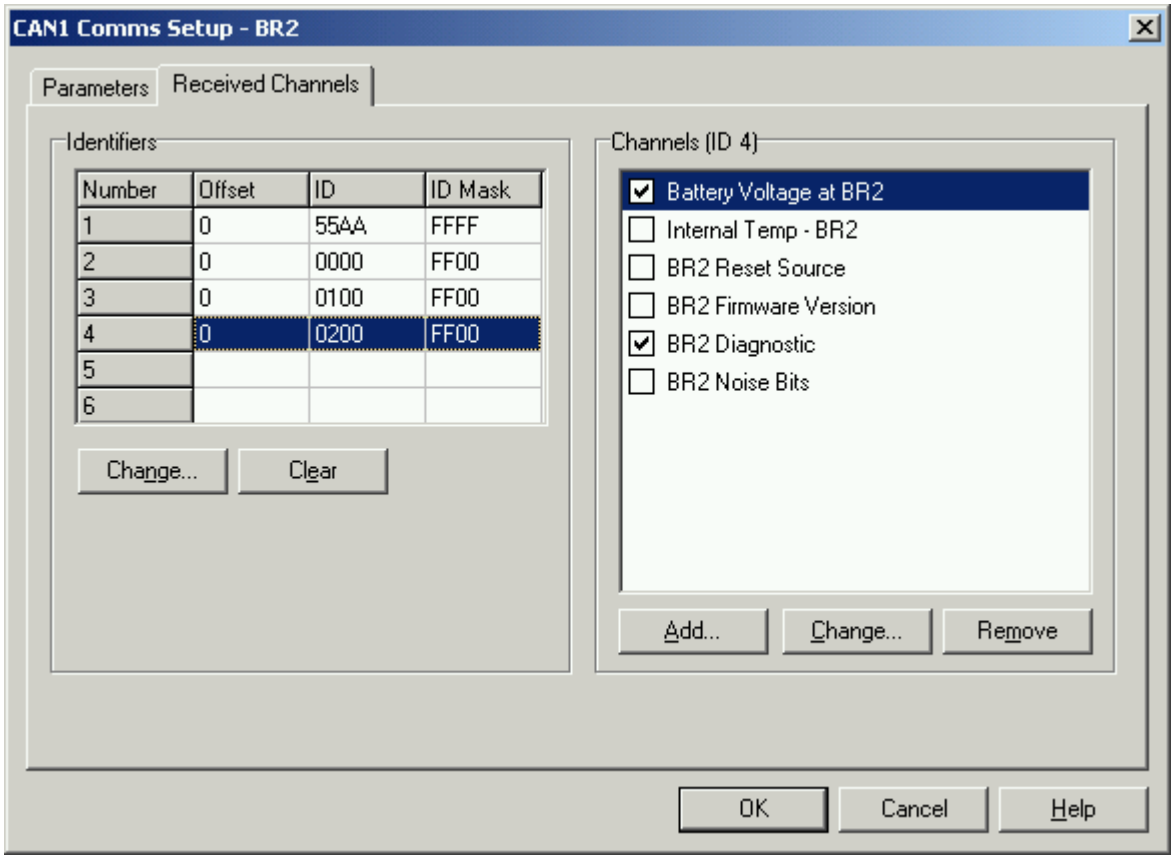

Fig 3: Received Channel Setup for the BR2 CAN template

Compound Messaging and Alignment

Each CAN packet includes 8 bytes of data. Any of these bytes can be used as an additional ID to increase the number of channels that are transmitted using one base address. While it is possible to use 2 bytes as a compound ID, this would only reduce the amount of data sent in each packet. While it is also possible to use any of the 8 bytes as the compound ID, conventionally byte 0 is used.

In the ADL it is possible to receive data from up to six different base addresses – making a total of 6x8 = 48 bytes of data.

Each of the 6 base addresses can have up to 6 compound id's within it, leaving 7 bytes of data per packet, but increasing the amount of possible data to 6x6x7 = 252 bytes.

Note: if a channel is only received on one of the compound id's, then its update rate will be less than if it was received on more than compound message, or on a non-compound message.

Fig 4: Setup for Compound Message id's

Offset

The byte position in the data packet that is being used as the compound identifier. This can be any of the bytes from 0 – 7, with 0 (the first byte) most commonly used.

Identifier

This is the hexadecimal value of the compound id that has been assigned to this group of channels/packet. Note that this is not related to the Base Address. When using a single byte id, it can have any (arbitrary) value between 0x00 and 0xFF. See Appendix D for a conversion table between decimal and hexadecimal.

Identifier Mask

Typically we are only using one byte as an identifier, while the ADL will automatically takes two bytes from a packet.[3] The Identifier Mask is used to 'screen' out one of the bytes as will often contain channel data.[4] This is achieved by performing a bitwise AND operation on the received data and the Identifier Mask. (see example below) See Appendix A for a description of the bitwise AND operation.

Alignment

The example below demonstrates the difference between 'Normal' and 'Word Swap' Alignment in regard to Compound Id's.

Byte Position/Offset	0	1	2	3	4	5	6	7
Normal Alignment (big endian)	Comp ID	Ch1 HB	Ch1 LB	Ch2 HB	Ch2 LB	Ch3 HB	Ch3 LB	Unused
Word Swap (little endian)	Comp ID	Ch1 LB	Ch1 HB	Ch2 LB	Ch2 HB	Ch3 LB	Ch3 HB	Unused
data	01	xx	yy	--	--	--	--	--

Fig 5: Example CAN data packet

Ch1 LB: Low data byte (least significant) of channel 1
Ch1 HB: High data byte (most significant) of channel 1, and so on for Channel's 2 and 3
xx and yy represent the data bytes for Channel 1

As all channels in the ADL are treated as 16 bit values, it will grab the first 2 bytes to check for the compound id, but we only wish to look at the first byte. The ID and ID Mask parameters allow us to do this. Both parameters are given as hex (base 16) values.

For the above example we have:

Alignment: Normal

Offset = 0 With an offset of 0, we are using the first byte in the packet (byte 0) as the compound identifier. This instructs the dash to take the first 2 bytes and read them as: 01xx

[3] All channels are treated as 16 bit values (2 bytes) for any calculations
[4] Without this, it is unlikely that the channel data would match the Identifier specified and few (if any) valid packets would be received

ID = 0100 01 is the id associated with the channels listed, while 00 indicates that we are ignoring the other byte. (Note that it is possible to have an id of 00)

ID Mask = FF00 The ID Mask is 'bitwise anded'[5] with the data before being matched against the ID. The result here is that the 'FF' passes the first (high) byte, while the '00' clears any data (the 'xx' or Ch1 LB) setting it to '00' to match the 00 in the ID above.

Alignment: Word Swap

Offset = 0 Again we are using the first byte in the packet as the compound id, but this time the ADL takes the first two bytes and reads them as: xx01 – hence the name 'word swap'

ID = 0001 The first byte is 00 indicating we will be ignoring this, while the second byte is 01 to match the id we are looking for.

ID Mask = 00FF The '00' will clear any data (Ch1 HB) setting the value to '00'.
The 'FF' passes the other byte unchanged for comparison with the ID

Change Comms Channel

Applies to both Single and Compound Message Types

Once the format of the data is determined, channels can be assigned to the incoming data. When a channel is added using the 'Add' button, or then modified by using 'Change' a dialog appears with the following parameters.

Channel: The received channel is selected from the master channel list.

Default Value: If there is an interruption to the datastream, or no data at all, then this is the channel value that will be used.

Offset: The 'distance' in bytes from the start of the data packet to the start of the channel. The first byte has an offset of zero.

Length: The length of the channel in bytes. All channels used in the ADL have a length of 2 bytes (except for error and status channels). A channel from another device that is only one byte long can be received and put into a 2 byte channel.

Channel Scaling:

The next parameters are used to scale the incoming 'raw' data so that it fits into the units and resolution of the ADL channel. The scaling is applied in the order: multiplier, divisor, then adder.

Multiplier: The raw data is multiplied by this value,

Divisor: Then divided by this value,

Adder: Then has this number added to it (can be negative).

NOTE: After this, the resulting figure is assigned to the channel selected. This will determine where the decimal point is placed. Up to this point the data is an integer value, it is only after the scaling is applied that the value is treated as having a decimal place (if any). Note that this is determined by the resolution of the channel and will affect the parameter values used.

For example: The base units of rpm in the ADL are Hertz, with a resolution of 0.1
An ECU might transmit Engine RPM in rpm with a resolution of 1 rpm.
1 Hz = 60 rpm, so to convert rpm to Hz divide by 60.

But, the channel resolution is 0.1Hz so the decimal place will be inserted one place in from the least significant digit. This means we need to multiply the data by 10. The result is a Multiplier of 10 and a Divisor of 60 (this can be reduced to a Multiplier of 1 and a Divisor of 6).

[5] see Appendix A

Bit Mask: This works the same way as the ‘Bit And’ function in ADL Channel Maths. Used to ‘screen’ the data so that only one (usually, though can be more in any combination) bits are captured

Signed: This is checked if the received channel uses the MSB (most significant bit) to indicate that the value is negative. A 2’s complement system is assumed (see Appendix B)

Comms Error Codes

The "Comms CAN * Diagnostic" and "Comms RS232 Diagnostic" channels can take the following values:

0	OK
1	PARITY
2	FRAMING
4	NOISE
8	OVERRUN
512	NO_DATA
1024	CHKSUM
4096	BUS_WARN
8192	BUS_OFF

Multiple errors are shown by error codes added together.
eg: 6 = noise + framing

FRAMING - the baud rate is probably wrong (or there is noise) as invalid RS232 data is being received

PARITY - the comms setup is wrong, or there is noise

NOISE - there is probably some noise

NO_DATA - a valid message header has not been found - either there is a wiring fault or comms is setup incorrectly (could be either end)

CHECKSUM - a valid message header has been found, but the checksum was wrong. If seen in combination with other errors there is noise. If only checksum errors occur there may be a software incompatibility between the ADL and the other device

BUS_WARN - there has been more than 96 errors on the CAN bus. Check wiring and termination resistors. The CAN bus may still be operational.

BUS_OFF - there have been more than 255 errors on the CAN bus. CAN communication is suspended when this error occurs. Check wiring and termination resistors. Check the CAN baud rate. Check CAN HI and CAN LO are correct (not swapped).

Unload comms errors:

(Errors that can occur when unloading log data from the ADL)

01h = timeout
02h = checksum
04h = framing error
08h = overrun
10h = min_char (not enough characters received by timeout)

Appendix A – Bitwise AND Operation

The results of a logical AND operation can be seen in the following 'truth' table.

inputs	0	1
0	0	0
1	0	1

In other words; 0 AND 0 = 0
1 AND 0 = 0 (0 AND 1 = 0)
1 AND 1 = 1

This operation takes two sixteen bit numbers and 'lines them up', then performs an AND operation between bits in the same position. Ie: most significant bit with most significant bit down to LSB and LSB.

For example:

1010 1100
AND 1011 0111
= 1010 0100

For the purposes of identifier masking, 'any value' AND 0 becomes 0 (clearing any data), while 'any value' AND 1 remains unchanged. (In binary 'any value' is either 1 or 0)

Appendix B – 2's Complement Numbering

There are two main methods used for specifying negative numbers in binary systems.

One way of doing this would be to say that if the MSB[6] has been set (ie: = 1), then the number is negative. Ie: instead of having a numerical weighting, this bit indicates the sign of the number, positive if 0, negative if 1.

The 2's complement system is slightly different. If the MSB has been set, then the weighting (or value) of this bit is subtracted from the rest of the number. As its value is always greater than the rest of the bits combined any number with the MSB set will be negative.

Using a four bit binary number as an example, ie: numbers from 0000 – 1111
The value of the MSB in decimal is '8' so in the 2's complement system this is read as '-8'.
This gives us a range of values from '-8' (1000) to '7' (0111).
For example: the number 1001 = -7 as it consists of '-8 + 1'.

For 16 bit calculations in the ADL, the MSB has a value of 32768, which is –32768 in 2's complement. This gives us a range of –32768 to 32767 for channel values in the ADL (ignoring the decimal place).

Appendix C – Bits and Bytes

Binary data consists of single 'bits' of information that have a value of either 0 or 1. In computer systems single bits are grouped together, either to represent characters or other information such as numbers (data).

A group of 8 bits makes one byte. The MoTeC ADL uses 16 bit channels, in other words, each channel is two bytes long.[7]

A byte can also be divided into 2 'nibbles' of four bits each. This is convenient as a four bit nibble can be represented by one digit in the hexadecimal numbering system.

This system is used in the setup of communications parameters when entering Compound Id's and Identifier Masks.

[6] Most Significant Bit
[7] Except for error channels, which are 1 byte long

Appendix D - Decimal → Binary → Hexadecimal Conversion

Decimal	Binary	Hex
0	0 (0000)	0
1	1 (0001)	1
2	10 (0010)	2
3	11 (0011)	3
4	100 (0100)	4
5	101 (0101)	5
6	110 (0110)	6
7	111 (0111)	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F
16	0001 0000	10
17	0001 0001	11