Professional Electronics
for Automotive and Motorsport

6 Repton Close | Basildon
Essex | SS13 1LE | United Kingdom
+44 (0) 1268 904124
info@liferacing.com
www.liferacing.com

LR CAN
User Manual

Document revision:
V1.1

Document Release Date:
2019-10-01

Document Author:
MH

Contents

1 Introduction

This document contains all of the Life Racing default datastream information and explains how to customise these settings for communication with other Life Racing products and custom CAN projects.

A Controller Area Network (CAN) is a communication standard most commonly seen in automotive applications to allow multiple controllers and devices to communicate on a single “bus” rather than individual wiring between each unit (for more information go to https://en.wikipedia.org/wiki/CAN_bus). All Life Racing controllers have at least 1 CAN bus. Life Racing CAN datastreams are always CAN2.0B using 11 bit identifiers with a configurable bus frequency.

Serial transmission is also used in some applications but is generally being replaced by CAN for most automotive uses (https://en.wikipedia.org/wiki/RS-232). All Life Racing ECUs are capable of transmitting RS232. Life Racing RS232 datastreams are always asynchronous serial at 38400/N/8/1.

2 CAN Fundamentals

2.1 Binary and Hexadecimal

A good understanding of Binary and Hexadecimal notation is helpful in understanding how CAN operates. A single unit of data is known as a bit (short for binary digit). A bit has a single binary value; either 0 or 1. A byte consists of 8 bits and has 256 combinations. A decimal value can be given to a byte by giving each bit position a place value twice as large as the one before. In this way it is possible to count by adding the active place values as demonstrated below with a 3bit array.

Bit Position:		2	1	0
Place Value:		**4**	**2**	**1**
Decimal	**0**	0	0	0
	1	0	0	1
	2	0	1	0
	3	0	1	1
	4	1	0	0
	5	1	0	1
	6	1	1	0
	7	1	1	1

Each added bit doubles the number of combinations giving a byte a range of 0-255 and two bytes (typically known as a “word”) a range of 0-65,535. As the notation of 16 characters is inconvenient, binary is often shortened using Hexadecimal. This groups every

4 bits (also known as nibbles) into a single character between 0 and F where the letters A-F represent the decimal values of 10-15.

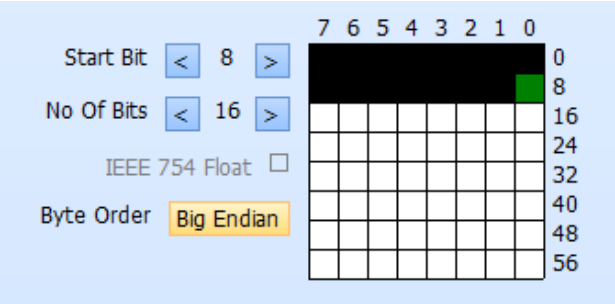
Bit Position:		7	6	5	4	3	2	1	0
Place Value:		**128**	**64**	**32**	**16**	**8**	**4**	**2**	**1**
Hex	**Dec**	(8)	(4)	(2)	(1)	8	4	2	1
07h	**7**	0	0	0	0	0	1	1	1
53h	**83**	0	1	0	1	0	0	1	1
E8h	**232**	1	1	1	0	1	0	0	0

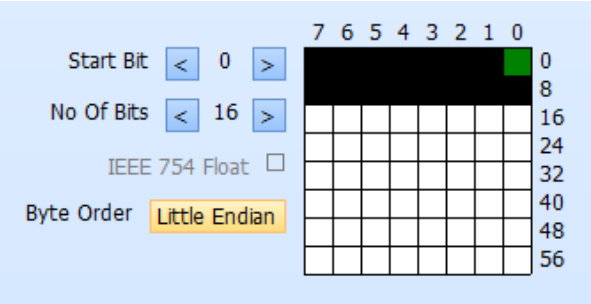
In order to represent a negative value, data can be marked as SIGNED. This changes how the bits are translated into decimal by swapping the sign on the final place value as below. This gives a byte the range of -128 to +127 and a 16bit word -32,768 to +32,767.

Bit Position:		7	6	5	4	3	2	1	0
Place Value:		**-128**	**64**	**32**	**16**	**8**	**4**	**2**	**1**
Decimal	**7**	0	0	0	0	0	1	1	1
	83	0	1	0	1	0	0	1	1
	-24	1	1	1	0	1	0	0	0

2.2 Basic Structure

A CAN frame consists of a user defined address or ID followed by control variables that describe the data, followed by the data itself and finally an error check and acknowledgement section. The data can contain up to 8 Bytes of data. The data may include several separate variables, split into any consecutive number of bits. As long as the receiving device knows this layout, it can be decoded. If a transmitted value crosses over more than one byte, it must be defined which byte to read first; that with the least significant bit (little endian/low byte first/Intel) or that with the most significant bit (big endian/high byte first/Motorola).





Bit Position:		1st Byte								2nd Byte							
Little Endian	21,475	128	64	32	16	8	4	2	1	13768	16384	8192	4096	2048	1024	512	256
		1	1	1	0	0	0	1	1	0	1	0	1	0	0	1	1
Big Endian	21,475	13768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1
		0	1	0	1	0	0	1	1	1	1	1	0	0	0	1	1

As well as identifying a frame, an address also sets a priority order for when two devices attempt to transmit at the same time. As a zero is dominant over a one, the lower the address, the higher the priority. The device transmitting a higher address will notice that it has been overruled and stop transmitting.

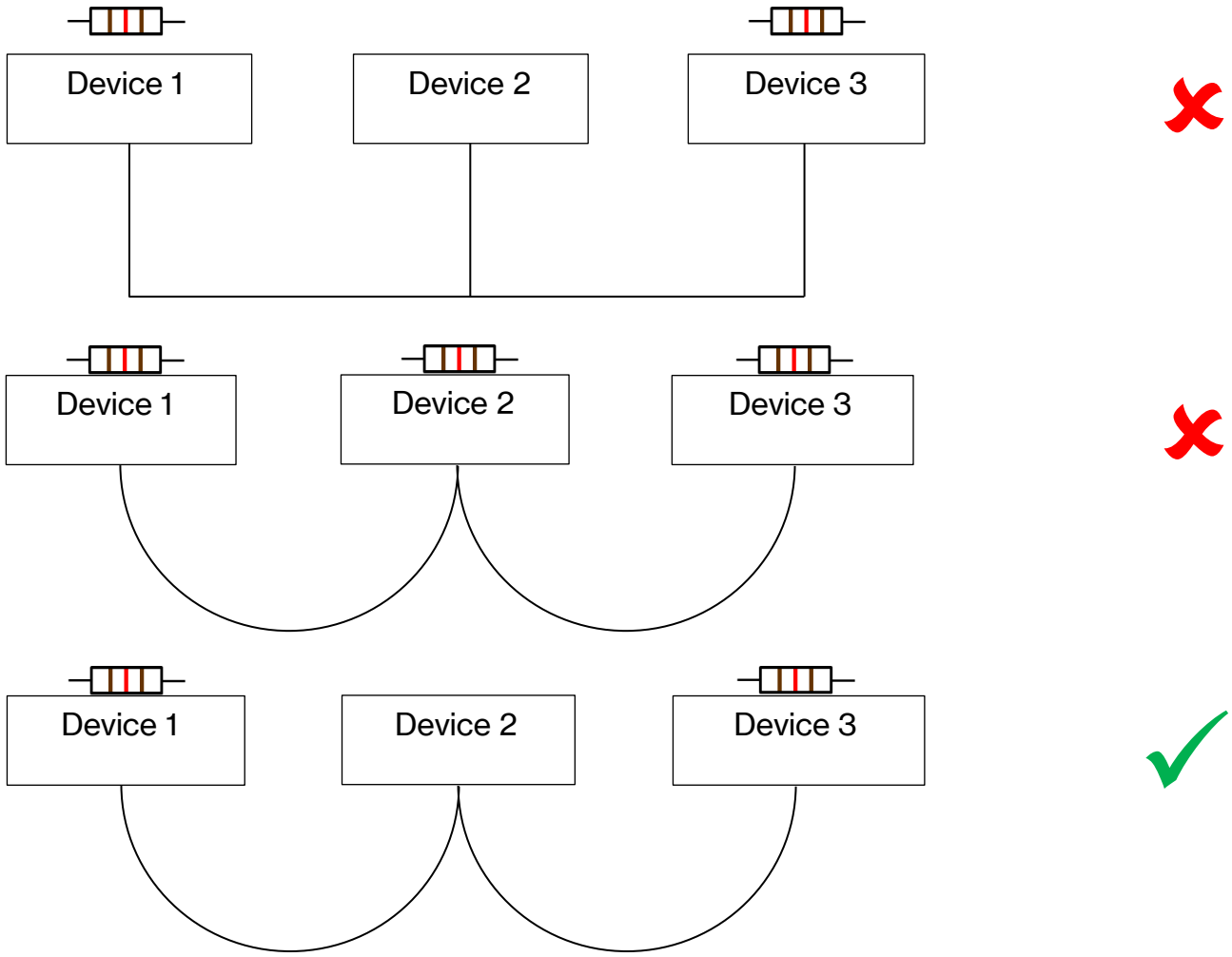
	ID bits											Rest of frame
	10	9	8	7	6	5	4	3	2	1	0	
Device 1 600h	0	1	0	0	1	0	1	1	0	0	0	...
Device 2 618h	0	1	0	0	1	1	STOPPED TRANSMITTING					
Resultant	0	1	0	0	1	0	1	1	0	0	0	...

2.3 Wiring

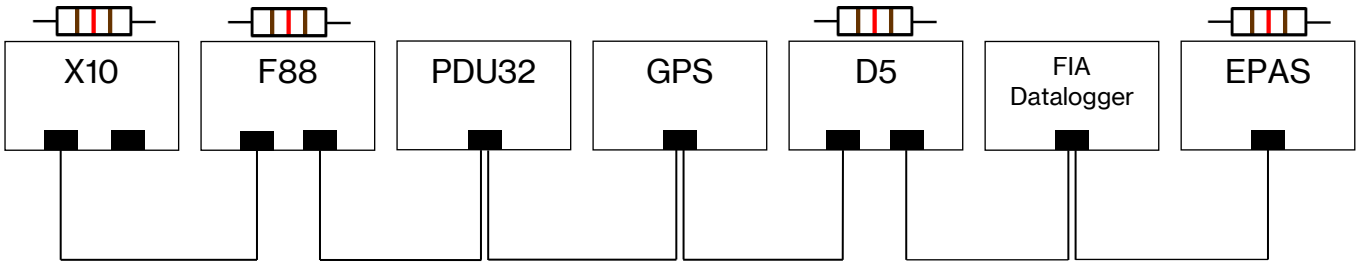
A CAN bus must be wired in a specific fashion for reliable communication. Wires should be a twisted pair with termination resistors at each end of the bus **only** (many devices have these built in so check with the supplier) and in a “daisy chain” layout.

Depending on the age and type of product, Life Racing controllers may have fixed termination, no termination or software selectable termination. Refer to their individual datasheets for this information. If a product without termination is used at the end of a bus, a 120Ω resistor should be fitted across the CAN wires as close to the pins as possible.

When connecting multiple devices, the CAN wires must go from device to device in a “daisy chain” layout and not have any “branches” to intermediary devices.



An example of a more complicated CAN system can be seen below.



The first CAN bus is a dedicated slave link between CAN2 on the master ECU (F88) and CAN1 on the slave expander (X10).

The second CAN bus is the main link for the F88 (CAN 1). The ECU is receiving PDU and GPS information as well as sending display and PDU control items. The D5 is able to convert all frames into a format required by the FIA logger.

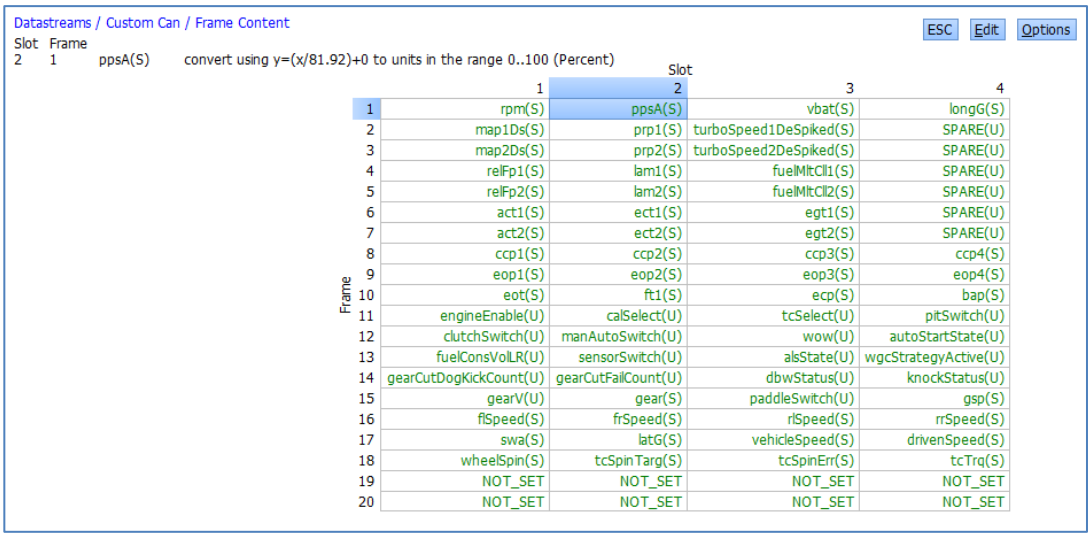
The third CAN bus has been assigned to non LR devices. The D5 is used to translate information and relay to and from the ECU on its other CAN link.

3 ECU

Life Racing ECUs have between 1 and 3 CAN buses available. Slave links only use CAN 2 and 3.

3.1 Transform Functions

Each item has its own fixed transform function which is listed in LifeCal. Highlighting a CAN frame item displays the conversion text on screen. The example below shows that to decode Pedal Position A (ppA) into engineering units a transform function of y=(x/81.92)+0 must be applied. Therefore to encode this item for CAN receive into the ECU the opposite is applied; a transform function of y=(x*81.92)+0 must be configured into the transmission. Item names with suffix _S are signed (+/-32000), those with suffix _U are unsigned (0-64000).



Frame \ Slot	1	2	3	4
1	rpm(S)	ppsA(S)	vbat(S)	longG(S)
2	map1Ds(S)	prp1(S)	turboSpeed1DeSpiked(S)	SPARE(U)
3	map2Ds(S)	prp2(S)	turboSpeed2DeSpiked(S)	SPARE(U)
4	relFp1(S)	lam1(S)	fuelMltCll1(S)	SPARE(U)
5	relFp2(S)	lam2(S)	fuelMltCll2(S)	SPARE(U)
6	act1(S)	ect1(S)	egt1(S)	SPARE(U)
7	act2(S)	ect2(S)	egt2(S)	SPARE(U)
8	ccp1(S)	ccp2(S)	ccp3(S)	ccp4(S)
9	eop1(S)	eop2(S)	eop3(S)	eop4(S)
10	eot(S)	ft1(S)	ecp(S)	bap(S)
11	engineEnable(U)	calSelect(U)	tcSelect(U)	pitSwitch(U)
12	clutchSwitch(U)	manAutoSwitch(U)	wow(U)	autoStartState(U)
13	fuelConsVolLR(U)	sensorSwitch(U)	alsState(U)	wgcStrategyActive(U)
14	gearCutDogKickCount(U)	gearCutFailCount(U)	dbwStatus(U)	knockStatus(U)
15	gearV(U)	gear(S)	paddleSwitch(U)	gsp(S)
16	flSpeed(S)	frSpeed(S)	rlSpeed(S)	rrSpeed(S)
17	swa(S)	latG(S)	vehicleSpeed(S)	drivenSpeed(S)
18	wheelSpin(S)	tcSpinTarg(S)	tcSpinErr(S)	tcTrq(S)
19	NOT_SET	NOT_SET	NOT_SET	NOT_SET
20	NOT_SET	NOT_SET	NOT_SET	NOT_SET

Enumerations can be viewed in the information panel of Life View with a particular channel highlighted.

```
gear (EVENT) samples(487)
No Units
UNKNOWN(0), REVERSE(1), NEUTRAL(2), FIRST(3), SECOND(4), THIRD(5), FOURTH(6),
FIFTH(7), SIXTH(8), SEVENTH(9), EIGHTH(10)
```

3.1.1 Unique Items

PDU Control Items

Outputs can be set to transmit over CAN as inputs for the PDU. These outputs should be assigned in the Pin Assignments as X:PDU CONTROL items.

By default, the PDU expects to see inputs on frames 710h and 711h in 8bit format. The ECU will transmit these frames in pairs, i.e. input 1&2 on a single 16bit word. Using default PDU settings, the ECU should transmit the following frame configuration:

	1	2	3	4
710h	pduCtl0102	pduCtl0304	pduCtl0506	pduCtl0708
711h	pduCtl0910	pduCtl1112	pduCtl1314	pduCtl1516

Error Flags

Bit flags transmit the status of up to 32 inputs across 2 16bit words; errorFlagsL and errorFlagsH. Each bit in these words represents an input where 0=OK and 1=Error. This may be a sensor failure or a CAN frame timeout depending on the input and its settings. What inputs are positioned where is set under Sensors / Error Flags, Warning Level, Limp, Trip / Error Flags Items Bit Assignments (0-15 for errorFlagsL and 16-31 for errorFlagsH).

Status Channels

Some status channels utilise bit flags to efficiently transmit the status of several relevant items in a single word. Contact Life Racing for more information.

3.2 CAN Receive

3.2.1 Generic Receive

Life Racing ECUs allows the receiving of defined and user defined sensors over CAN.

The datastream is always CAN2.0B using 11 bit identifiers with a configurable bus frequency of 1MHz/500Hz/250Hz (1Mbit/500Kbit/250Kbit per second). Data frames are always 8 bytes, consisting of four 16-bit quantities or eight 8-bit quantities sent high byte first (MSB). Custom receive settings can be found under Datastreams / Generic CAN Receive.

Up to 12 different frames can be defined allowing for 32 16-bit quantities and 32 8-bit quantity slots. Each frame can be received on any CAN bus not used by a slave, has a configurable CAN identifier and individual timeout settings. Behavior on timeout is programmable per sensor. By default, frames 640-643h and 646-649h are used for 16bit slot “A” frames and 644-645h and 64A-64Bh for 8bit slot “B” frames. Each incoming item must conform to its assigned Life Racing transform function unless it is a user definable sensor.

With generic receive enabled, the receive frames will be selectable in IO Configuration / Pin Assignments as virtual pins named in the format X:CAN RECEIVE A(B) #XX. Most inputs, including user definable sensors, can be received over CAN. Only critical or engine synchronous inputs such as Crank and Cam cannot be received using this method.

3.2.2 LR PDU CAN Receive

Up to three PDU datastreams can be received by an LR ECU. Each can be received on a selectable CAN bus with individual base identifiers and timeouts. Monitoring items will be labelled as PDUA, B or C. PDU datastream must be set to 0.2A resolution.

3.2.3 LR GPS CAN Receive

To receive information from an LR GPS, select the relevant unit from Datastreams / LR GPS CAN Receive / Receive LR GPS CAN Frames and chose the connected CAN bus. If GPS speed information is lost, the default value can be specified as zero speed or maximum reading depending on preference and if it is used as a variable in any strategies.

3.2.4 Other common devices

Some common devices with more complex structures are supported. These are split into their own branches under Datastreams where they can be individually toggled on and off and any relevant settings can be changed. Currently supported devices include:

- Basic GPS
- Bosch Accelerometer (YRS3 or MM5.10)
- Bosch ABS (M4 or M5)
- Texense IB6C Accelerometer
- MagCanica Torque Sensor
- Megaline E-Shift

3.2.5 Custom Projects

For more complex CAN structures and requirements, contact your local distributer to find out what Life Racing can do for your project.

3.3 CAN Transmit

Life Racing ECUs allows the calibration engineer to configure a CAN transmission datastream with flexible identifiers, transmission rates and content any CAN bus not used by a slave.

The datastream is always CAN2.0B using 11 bit identifiers with a configurable bus frequency of 1MHz/500Hz/250Hz (1Mbit/500Kbit/250Kbit per second). Data frames are always 8 bytes, consisting of four 16-bit quantities sent high byte first (MSB).

Up to 40 different frames can be defined (20 frames only pre V1.557.1). Each frame has configurable CAN identifier, transmission rate (up to 100Hz) and four transmitted quantities (selectable from all monitorable and loggable items within the ECU). Item names with suffix _S are signed (+/-32000), those with suffix _U are unsigned (0-64000).

The default template can be altered to allow for custom content using custom addresses and frequencies. The settings for this can be found under Datastreams / Generic CAN transmit.

3.3.1 Default Datastream

The default datastream defines 18 of the possible 40 frames. Identifiers are 600h through 611h. Transmission frequencies vary from 5Hz to 50Hz.

		Slot 1		Slot 2		Slot 3		Slot 4	
		Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
Frame 1 600h 50Hz	Content	rpm_S		ppsA_S		Vbat_S		longG_S	
	Unit	rpm		%		V		G	
	Transform	None		Divide by 81.92		Divide by 1000		Divide by 1000	
Frame 2 601h 50Hz	Content	map1Ds_S		prp1_S		turboSpeed1DeSpiked_S		SPARE	
	Unit	mBar		mBar		kRpm		-	
	Transform	None		None		Divide by 100		-	

		Slot 1		Slot 2		Slot 3		Slot 4	
		Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
Frame 3 602h 50Hz	Content	map2Ds_S		prp2_S		turboSpeed2DeSpiked_S		SPARE	
	Unit	mBar		mBar		kRpm		-	
	Transform	None		None		Divide by 100		-	
Frame 4 603h 10Hz	Content	relFp1_S		lam1_S		fuelMltCll1_S		SPARE	
	Unit	mBar		lambda		-		-	
	Transform	None		Divide by 1000		Divide by 4096		-	
Frame 5 604h 10Hz	Content	relFp2_S		lam2_S		fuelMltCll2_S		SPARE	
	Unit	mBar		lambda		-		-	
	Transform	None		Divide by 1000		Divide by 4096		-	
Frame 6 605h 5Hz	Content	act1_S		ect1_S		egt1_S		SPARE	
	Unit	degC		degC		degC		-	
	Transform	Divide by 10		Divide by 10		Divide by 10		-	
Frame 7 606h 5Hz	Content	act2_S		ect2_S		egt2_S		SPARE	
	Unit	degC		degC		degC		-	
	Transform	Divide by 10		Divide by 10		Divide by 10		-	
Frame 8 607h 5Hz	Content	ccp1_S		ccp2_S		ccp3_S		ccp4_S	
	Unit	mBar		mBar		mBar		mBar	
	Transform	None		None		None		None	
Frame 9 608h 10Hz	Content	eop1_S		eop2_S		eop3_S		eop4_S	
	Unit	mBar		mBar		mBar		mBar	
	Transform	None		None		None		None	
Frame 10 609h 5Hz	Content	eot_S		ft1_S		ecp_S		bap_S	
	Unit	degC		degC		mBar		mBar	
	Transform	Divide by 10		Divide by 10		None		None	
Frame 11 60Ah 5Hz	Content	engineEnable_U		calSelect_U		tcSelect_U		pitSwitch_U	
	Unit	-		-		-		-	
	Transform	Enumeration*		Add 1		Add 1		Enumeration*	

Frame 12 60Bh 5Hz	Content	clutchSwitch_U	manAutoSwitch_U	wow_U	autoStartDate_U
	Unit	-	-	-	-
	Transform	Enumeration*	Enumeration*	Enumeration*	Enumeration*
Frame 13 60Ch 5Hz	Content	fuelConsVolLR_U	sensorSwitch_U	alsState_U	wgcStrategyActive_U
	Unit	Litres	-	-	-
	Transform	Divide by 10	Enumeration*	Enumeration*	Enumeration*
Frame 14 60Dh 5Hz	Content	gearCutDogKickCount_	gearCutFailCount_U	dbwStatus_U	knockStatus_U
	Unit	-	-	-	-
	Transform	None	None	Bit Flags†	Bit Flags†
Frame 15 60Eh 50Hz	Content	gearV_U	gear_S	paddleSwitch_U	gsp_S
	Unit	V	-	-	-
	Transform	Divide by 1000	Enumeration*	Enumeration*	None
Frame 16 60Fh 50Hz	Content	flSpeed_S	frSpeed_S	rlSpeed_S	rrSpeed_S
	Unit	kph	kph	kph	kph
	Transform	Multiply by 0.036	Multiply by 0.036	Multiply by 0.036	Multiply by 0.036
Frame 17 610h 50Hz	Content	swa_S	latG_S	vehicleSpeed_S	drivenSpeed_S
	Unit	deg	G	kph	kph
	Transform	Divide by 32	Divide by 1000	Multiply by 0.036	Multiply by 0.036
Frame 18 611h 50Hz	Content	wheelSpin_S	tcSpinTarg_S	tcSpinErr_S	tcTrq_S
	Unit	%	%	%	%
	Transform	Divide by 10.24	Divide by 10.24	Divide by 10.24	Divide by 10.24

*Enumeration items below:

engineEnable
0=OK, 1=SW OFF, 2=SLAVE1_COMMS_OUT, 3=SLAVE2_COMMS_OUT, 4=VBAT_HIGH, 5=EOP_START, 6=CAN_ENABLE_OFF, 7=TPS_START, 8=AUTO_START, 9=FP_START, 10=DIP_START, 11=SENSOR_WARNING_LEVEL, 12=PDU_ISSUE, 13=AUTO_IN_R/D, 100=EOP_TRIP, 101=CCP_TRIP, 102=EOT_TRIP, 103=ECT_TRIP, 104=FP_TRIP, 105=DATE/TIME_TRIP, 106=VBAT_TRIP, 107=LEAN_TRIP
pitSwitch
0=OFF, 1=ON
clutchSwitch
0=OFF, 1=ON
manAutoSwitch
0=MANUAL, 1=AUTO

wow
0=OFF, 1=ON
autoStartState
0=OFF, 1=ENABLED, 2=ARMED, 3=STARTING, 4=FAILED, 5=CRANKING, 6=CLUTCH
sensorSwitch
0=OFF, 1=ON
alsState
0=OFF, 1=START, 2=ON, 3=SW_SHUTDOWN, 4=TPS_SHUTDOWN, 5=TPS_TIMEOUT, 100=DISABLED
wgcStrategyActive
0=OLD, 1=WGP-BASED, 2=STANDARD
gear
0=U, 1=R, 2=N, 3=1, 4=2, 5=3, 6=4, 7=5, 8=6, 9=7, 10=8
paddleSwitch
0=NONE, 1=DOWN, 2=UP, 3=BOTH

†Bit Flag items below:

dbwStatus	B0=PPS, B1=TPS1, B2=TPS2, B3=DBW1, B4=DBW2
knockStatus	B0=cyl1 B1=cyl2 etc

3.4 Slave/Expander

Slave ECUs and expander modules communicate with the master ECU on dedicated CAN buses.
An X10 expander module is already setup to operate as a slave but an ECU needs to be calibrated under IO Configuration / Operate As Slave ECU. If a slave is considered critical, Engine Enable can be disabled without connection under IO Configuration / Engine Operation Requires Slave 1/2 (If Used).
In Pin Assignments, assign additional inputs and outputs to Slave1 for CAN2 and Slave2 for CAN3. These inputs and outputs can then be configured as if they were directly connected to the master ECU. When a slave input or output is configured, the corresponding CAN bus will be claimed as a slave link so no other custom CAN may be used on this bus.

4 PDU

4.1 PDU Datastream

The PDU continually transmits input and output states and currents on CAN at up to 100Hz from a software definable “Base ID”. The channel definitions and data byte layout is below. Multibyte quantities are sent high byte first.

		Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
Frame 1 700h	Content	Hard Input States				Soft Input States			
	Unit	-				-			
	Transform	Bit Flags[1]				Bit Flags[1]			
Frame 2 701h	Content	opState01	opState02	opState03	opState04	opState05	opState06	opState07	opState08
	Unit	-	-	-	-	-	-	-	-
	Transform	Enumeration[2]	Enumeration[2]	Enumeration[2]	Enumeration[2]	Enumeration[2]	Enumeration[2]	Enumeration[2]	Enumeration[2]
Frame 3 702h	Content	opState09	opState10	opState11	opState12	opState13	opState14	opState15	opState16
	Unit	-	-	-	-	-	-	-	-
	Transform	Enumeration[2]	Enumeration[2]	Enumeration[2]	Enumeration[2]	Enumeration[2]	Enumeration[2]	Enumeration[2]	Enumeration[2]
Frame 4 703h	Content	opState17	opState18	opState19	opState20	opState21	opState22	opState23	opState24
	Unit	-	-	-	-	-	-	-	-
	Transform	Enumeration[2]	Enumeration[2]	Enumeration[2]	Enumeration[2]	Enumeration[2]	Enumeration[2]	Enumeration[2]	Enumeration[2]
Frame 5 704h	Content	opState25	opState26	opState27	opState28	opState29	opState30	opState31	opState32
	Unit	-	-	-	-	-	-	-	-
	Transform	Enumeration[2]	Enumeration[2]	Enumeration[2]	Enumeration[2]	Enumeration[2]	Enumeration[2]	Enumeration[2]	Enumeration[2]
Frame 6 705h	Content	current01	current02	current03	current04	current05	current06	current07	current08
	Unit	Amps	Amps	Amps	Amps	Amps	Amps	Amps	Amps
	Transform	Divide by 2	Divide by 2	Divide by 2	Divide by 2	Divide by 2	Divide by 2	Divide by 2	Divide by 2
Frame 7 706h	Content	current09	current10	current11	current12	current13	current14	current15	current16
	Unit	Amps	Amps	Amps	Amps	Amps	Amps	Amps	Amps
	Transform	Divide by 2	Divide by 2	Divide by 2	Divide by 2	Divide by 2	Divide by 2	Divide by 2	Divide by 2
Frame 8 707h	Content	current17	current18	current19	current20	current21	current22	current23	current24
	Unit	Amps	Amps	Amps	Amps	Amps	Amps	Amps	Amps
	Transform	Divide by 2	Divide by 2	Divide by 2	Divide by 2	Divide by 2	Divide by 2	Divide by 2	Divide by 2

		Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
Frame 9 708h	Content	current25	current26	current27	current28	current29	current30	current31	current32
	Unit	Amps	Amps	Amps	Amps	Amps	Amps	Amps	Amps
	Transform	Divide by 2	Divide by 2	Divide by 2	Divide by 2	Divide by 2	Divide by 2	Divide by 2	Divide by 2
Frame 10 709h	Content	Board Temperature		Battery Voltage		onTime			
	Unit	degC		Volts		Seconds			
	Transform	Divide by 10		Divide by 1000		Divide by 100			
Frame 11 70Ah	Content	An 01 Voltage	An 02 Voltage	An 03 Voltage	An04 Voltage	totalCurrent		Flags[3]	SPARE
	Unit	Volts	Volts	Volts	Volts	Amps		-	-
	Transform	Divide by 5	Divide by 5	Divide by 5	Divide by 5	Divide by 2		Bit Flags	None
Frame 12 70Bh	Content	opState33	current33	opState34	current34	opState35	current35	opState36	current36
	Unit	-	Amps	-	Amps	-	Amps	-	Amps
	Transform	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2
Frame 13 70Ch	Content	opState37	current37	opState38	current38	opState39	current39	opState40	current40
	Unit	-	Amps	-	Amps	-	Amps	-	Amps
	Transform	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2
Frame 14 70Dh	Content	opState41	current41	opState42	current42	opState43	current43	opState44	current44
	Unit	-	Amps	-	Amps	-	Amps	-	Amps
	Transform	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2
Frame 15 70Eh	Content	opState45	current45	opState46	current46	opState47	current47	opState48	current48
	Unit	-	Amps	-	Amps	-	Amps	-	Amps
	Transform	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2

Frame 10 not sent by pre v1.4.4 main code; sensors not present in pre v2.0 hardware
Frame 11 and “onTime” not sent by pre v2.11.4 main code

(1) Input State Bit Flags (Most Significant Bit is input 32, Least Significant Bit is input
0 OFF
1 ON

(2) Output State Enumeration:
0 OFF
1 ON
2 INRUSH
3 ALARM
100 SHORT
101 HIGHTRIP
102 INRUSHTRIP
103 LOWTRIP
200 TEAMTRIP

(3) Frame 11 Flags determine current resolution:
0 0.5A (default)
1 0.2A
With 0.2A resolution, all transformations become **Divide by 5**.

4.2 PDU Datastream v2

The PDU datastream format has been updated as of firmware version **2.15.2** to account for the higher number of outputs and functionality found in the newest generation of PDUs. The PDU continually transmits input and output states and currents on CAN at up to 100Hz from a software definable “Base ID”. The channel definitions and data byte layout is below. Multibyte quantities are sent high byte first.

		Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
Frame 1 700h Base + 0	Content	Board Temperature 1		Board Temperature 2		Power Supply Voltage		Logic Supply Voltage	
	Unit	degC		degC		Volts		Volts	
	Transform	Divide by 10		Divide by 10		Divide by 1000		Divide by 1000	
Frame 2 701h Base + 1	Content	Total Current		SPARE	SPARE	SPARE	SPARE	SPARE	Flags[3]
	Unit	Amps		-	-	-	-	-	-
	Transform	Divide by 2		None	None	None	None	None	Bit Flags
Frame 3 702h Base + 2	Content	Validated Hard Input States							
	Unit	-							
	Transform	Bit Flags[1]							
Frame 4 704h Base + 4	Content	Validated Soft Input States							
	Unit	-							
	Transform	Bit Flags[1]							
Frame 5 706h Base + 6	Content	Soft Output States							
	Unit	-							
	Transform	Bit Flags[1]							
Frame 6 708h Base + 8	Content	AN01V	AN02V	AN03V	AN04V	AN05V	AN06V	AN07V	AN08V
	Unit	Volts	Volts	Volts	Volts	Volts	Volts	Volts	Volts
	Transform	Divide by 50	Divide by 50	Divide by 50	Divide by 50	Divide by 50	Divide by 50	Divide by 50	Divide by 50
Frame 7 709h Base + 9	Content	AN09V	AN10V	AN11V	AN12V	AN13V	AN14V	AN15V	AN16V
	Unit	Volts	Volts	Volts	Volts	Volts	Volts	Volts	Volts
	Transform	Divide by 50	Divide by 50	Divide by 50	Divide by 50	Divide by 50	Divide by 50	Divide by 50	Divide by 50
Frame 8 70Ch Base + 12	Content	OP State 01	Current 01	OP State 02	Current 02	OP State 03	Current 03	OP State 04	Current 04
	Unit	-	Amps	-	Amps	-	Amps	-	Amps
	Transform	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2

		Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
Frame 9 70Dh Base + 13	Content	OP State 05	Current 05	OP State 06	Current 06	OP State 07	Current 07	OP State 08	Current 08
	Unit	-	Amps	-	Amps	-	Amps	-	Amps
	Transform	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2
Frame 10 70Eh Base + 14	Content	OP State 09	Current 09	OP State 10	Current 10	OP State 11	Current 11	OP State 12	Current 12
	Unit	-	Amps	-	Amps	-	Amps	-	Amps
	Transform	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2
Frame 11 70Fh Base + 15	Content	OP State 13	Current 13	OP State 14	Current 14	OP State 15	Current 15	OP State 16	Current 16
	Unit	-	Amps	-	Amps	-	Amps	-	Amps
	Transform	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2
Frame 12 710h Base + 16	Content	OP State 17	Current 17	OP State 18	Current 18	OP State 19	Current 19	OP State 20	Current 20
	Unit	-	Amps	-	Amps	-	Amps	-	Amps
	Transform	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2
Frame 13 711h Base + 17	Content	OP State 21	Current 21	OP State 22	Current 22	OP State 23	Current 23	OP State 24	Current 24
	Unit	-	Amps	-	Amps	-	Amps	-	Amps
	Transform	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2
Frame 14 712h Base + 18	Content	OP State 25	Current 25	OP State 26	Current 26	OP State 27	Current 27	OP State 28	Current 28
	Unit	-	Amps	-	Amps	-	Amps	-	Amps
	Transform	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2
Frame 15 713h Base + 19	Content	OP State 29	Current 29	OP State 30	Current 30	OP State 31	Current 31	OP State 32	Current 32
	Unit	-	Amps	-	Amps	-	Amps	-	Amps
	Transform	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2
Frame 16 714h Base + 20	Content	OP State 33	Current 33	OP State 34	Current 34	OP State 35	Current 35	OP State 36	Current 36
	Unit	-	Amps	-	Amps	-	Amps	-	Amps
	Transform	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2
Frame 17 715h Base + 21	Content	OP State 37	Current 37	OP State 38	Current 38	OP State 39	Current 39	OP State 40	Current 40
	Unit	-	Amps	-	Amps	-	Amps	-	Amps
	Transform	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2
Frame 18 716h Base + 22	Content	OP State 41	Current 41	OP State 42	Current 42	OP State 43	Current 43	OP State 44	Current 44
	Unit	-	Amps	-	Amps	-	Amps	-	Amps
	Transform	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2

		Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
Frame 19 717h Base + 23	Content	OP State 45	Current 45	OP State 46	Current 46	OP State 47	Current 47	OP State 48	Current 48
	Unit	-	Amps	-	Amps	-	Amps	-	Amps
	Transform	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2
Frame 20 718h Base + 24	Content	OP State 49	Current 49	OP State 50	Current 50	OP State 51	Current 51	OP State 52	Current 52
	Unit	-	Amps	-	Amps	-	Amps	-	Amps
	Transform	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2
Frame 21 719h Base + 25	Content	OP State 53	Current 53	OP State 54	Current 54	OP State 55	Current 55	OP State 56	Current 56
	Unit	-	Amps	-	Amps	-	Amps	-	Amps
	Transform	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2
Frame 22 71Ah Base + 26	Content	OP State 57	Current 57	OP State 58	Current 58	OP State 59	Current 59	OP State 60	Current 60
	Unit	-	Amps	-	Amps	-	Amps	-	Amps
	Transform	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2
Frame 23 71Bh Base + 27	Content	OP State 61	Current 61	OP State 62	Current 62	OP State 63	Current 63	OP State 64	Current 64
	Unit	-	Amps	-	Amps	-	Amps	-	Amps
	Transform	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2	Enumeration[2]	Divide by 2
Frame 24 71Eh Base + 30	Content	LS State 01	LS State 02	LS State 03	LS State 04	LS State 05	LS State 06	LS State 07	LS State 08
	Unit	-	-	-	-	-	-	-	-
	Transform	Enumeration[2]	Enumeration[2]	Enumeration[2]	Enumeration[2]	Enumeration[2]	Enumeration[2]	Enumeration[2]	Enumeration[2]
Frame 25 71Fh Base + 31	Content	LS State 09	LS State 10	LS State 11	LS State 12	LS State 13	LS State 14	SPARE	SPARE
	Unit	-	-	-	-	-	-	-	-
	Transform	Enumeration[2]	Enumeration[2]	Enumeration[2]	Enumeration[2]	Enumeration[2]	Enumeration[2]	None	None

(1) Input and output state Bit Flags (Most Significant Bit is input/output 64, Least Significant Bit is input/output 1):

0 OFF
1 ON

(2) Output State Enumeration:

0 OFF
1 ON
2 INRUSH
3 ALARM
100 HARDTRIP
101 HIGHTRIP
102 INRUSHTRIP
103 LOWTRIP
104 PWRVTRIP
105 BTTRIP
200 TEAMTRIP

(3) Frame 2 Flags determine current resolution:

0 0.5A (default)
1 0.2A

With 0.2A resolution, all current transformations become **Divide by 5**.

4.3 I/O

Soft inputs and outputs communicate using 8bit items. Each frame can contain 8 inputs or 8 outputs with customisable addresses. The default addresses range from 710..717h in numerical order for up to 64 inputs and 718..71Fh in numerical order for up to 64 outputs. These addresses can be altered under Cal, Communications. Enabling keypads reduces the number of inputs and outputs available as some become reserved.
The input byte is considered as signed; zero means turn the soft input off, a positive value means turn the soft input on, and a negative value means no change.

Soft inputs can be received from a Life Racing ECU using PDU Control items. These outputs should be assigned in the *Pin Assignments* as X:PDU CONTROL items.

By default, the PDU expects to see inputs on frames 710h and 711h in 8bit format (this is moved to 730h and 731h for datastream V2). The ECU will transmit these frames in pairs, i.e. input 1&2 on a single 16bit word. Using default PDU settings, the ECU should transmit the following frame configuration:

	1	2	3	4
710h	pduCtl0102	pduCtl0304	pduCtl0506	pduCtl0708
711h	pduCtl0910	pduCtl1112	pduCtl1314	pduCtl1516

4.4 Switch Panel

Check the Switch Panel boxes in the Cal, Communications dialogue to enable switch inputs. This will reserve some soft I/O and allow the assigning of inputs and outputs to Life Racing keypad inputs and LED feedback. If the Grayhill Switch Panels box is also checked, the settings will be changed for compatibility with CANopen Grayhill devices.

4.5 Fault Reset

Receipt of any valid frame with the correct identifier (set under Cal, Communications) is considered identical to a suitable duration short-to-ground of the fault reset input pin.

5 Dash

The dash units and MCH have the most flexible CAN configuration options and can therefore often be used as a central CAN hub, converting information from one format into another by utilising 2 CAN buses, maths functions and flexible frame content.
Each CAN bus can have independently set baud rates and termination. Baud rate options are 100Kbit, 125Kbit, 200Kbit, 250Kbit, 500Kbit and 1Mbit.

5.1 Receive from LR

5.1.1 ECU

To receive CAN information from a Life Racing ECU, Life Dash can extract the CAN information from the ECU calibration file. Under the 'CANrx' tab, select Add Frame / LRCan from .LRC, locate the ECU calibration and select the CAN bus. This will create all items in the datastream with the correct units and scaling. A preferred unit, name, gauge scale and alarm values can be customised under the 'Items' tab.
If the ECU and dash are connected by Ethernet, selecting LRCAN by Ethernet will move the datastream to Ethernet, relaxing the traffic on the CAN bus.

5.1.2 PDU

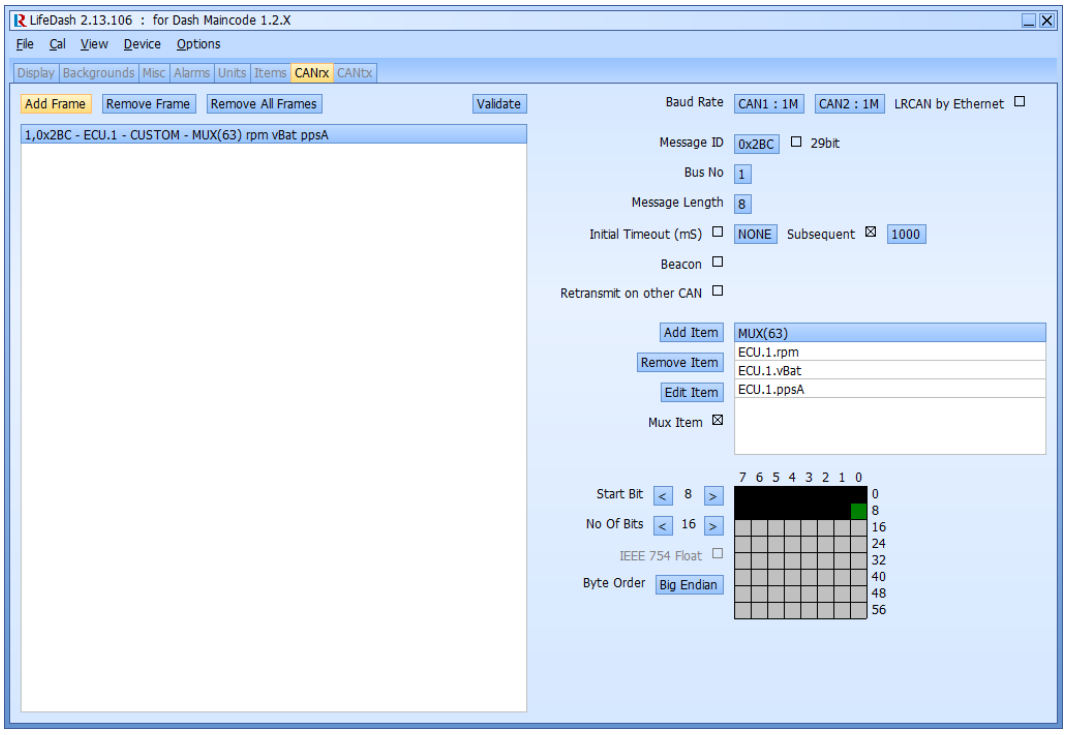
To receive CAN information from a Life Racing PDU, Life Dash can automatically setup the default datastream when given the base ID. Under the 'CANrx' tab, select Add Frame / LRPdu and select the model, base ID and CAN bus. This creates all PDU items and enables the PDU specific dash pages. The dash will automatically detect the current resolution from the 'flags' item (pdu v2.11.4 required).
PDU soft outputs can also be automatically added by selecting Add Frame / PDU Soft i/p with a standard PDU frame highlighted. Provide the CAN address and input range and these will be added to the previously highlighted PDU.
Similarly, a keypads soft outputs can be added by selecting Add Frame / SWP Soft i/p

5.1.3 GPS

To receive CAN information from a Life Racing GPS unit select Add Frame / LRGps and select the desired unit. Assuming default CAN settings, all GPS items will be added. Note that to use GPS position for beacons, the special items "LATITUDE" and "LONGITUDE" must be defined.

5.2 Custom Receive

For any other device, frames can be completely customised to the correct settings including mux items and irregular bit lengths. Select Add Frame / Custom to create a blank frame and follow the instructions for ID and device. A variety of settings can be changed using the options on the right.



General

Message ID:	Frame identifier
29bit:	Allows 29bit addresses to be used.
Bus no:	Which bus the frame will be received on. CAN 1 or CAN 2.
Length:	Frame size in bytes.
Initial Timeout:	Time after power up allowed before timeout error.
Subsequent:	Time between received frames allowed before timeout error.
Beacon:	Ticking this box will result in a beacon event trigger when this frame is received, regardless of content.

Frame content

Add Item:	Add an item to the selected frame.
Remove Item:	Delete the selected item.
Edit Item:	Select associated item from the list (items must first be added in the 'Items' tab).
Mux Item:	Ticking this box makes the selected item an ID item. Assigning different ID values to frames allows them to share message IDs.

Frame Layout

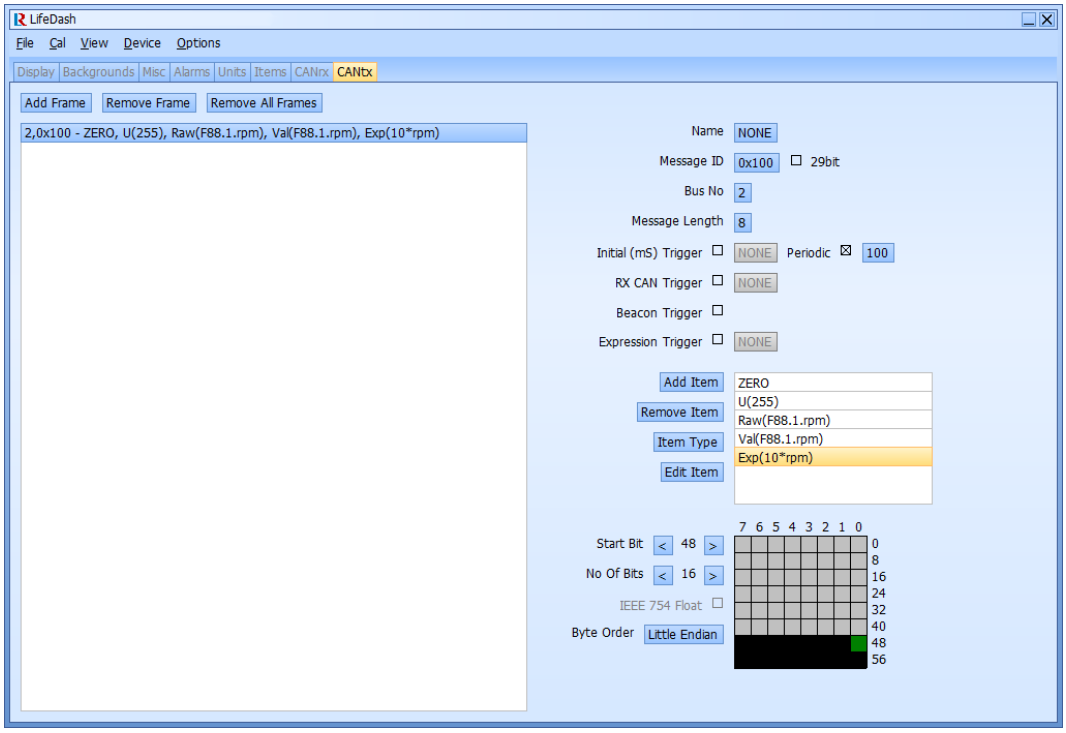
Frame:	Use the Start Bit and No Of Bits controls or drag across the grid to choose item size and location within the frame.
IEEE 754 Float:	Tick this box to allow floating bit points.
Byte Order:	Choose between big and little endian byte order for each item.

Validate

Pressing the ***Validate*** button will perform a check for any validation errors in the active CAN frames such as repeated IDs. A message will only be displayed if a conflict is found.

5.3 Transmit

Life Racing Display units also have fully flexible CAN transmission capabilities. Selecting Add Frame under the 'CANtx' tab creates a blank frame. The options on the right can then be used to fill and configure the frame.



General

Name:	Optional name of frame for easy identification in list
Message Id:	Frame identifier
29bit:	Allows 29bit addresses to be used.
Bus no:	Which bus the frame will be transmitted on. CAN1 or CAN 2
Message Length:	Length of frame in bytes

Triggers

Initial (mS) Trigger:	Time to wait before transmitting first frame
Periodic:	Time period between each transmission
RX CAN Trigger:	Transmit when selected frame is received
Beacon Trigger:	Transmit when beacon is triggered
Expression Trigger:	Transmit when a maths expression is true

Frame content

Add Item: Add an item to the selected frame.

Remove Item: Delete the selected item.

Item Type: Choose between:

- ZERO default blank,
- UNSIGNED numerical value, maximum dependent on No of bits,
- MII-RAW unscaled item values
- MII-VAL scaled item values, rounded to the nearest integer
- EXPRESSION mathematical expression

Edit Item: Open the dialogue box for the selected item type. Raw and Val item types will allow selection from the list of items created in the 'Items' tab.

Frame Layout

Frame: Use the Start Bit and No Of Bits controls or drag across the grid to choose item size and location within the frame.

IEEE 754 Float: Tick this box to allow floating bit points.

Byte Order: Choose between big and little endian byte order for each item.

Any frame received can be retransmitted on the opposite bus in the exact same format by selecting 'Retransmit on other CAN' under the 'CANrx' tab. This allows two devices on different CAN buses to communicate through the dash.

6 GPS

6.1 Datastream

By default, both the GPS-A10 and the GPS-AG50 transmit information at a base ID of 680h. For communication with a Life Racing ECU, this should not be changed. The ECU calibration should have the map under Datastreams / LR GPS CAN Receive / Receive LR GPS CAN Frames set to the relevant device.

		Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
GPS Position 680h 50Hz	Content	GPS latitude_S				GPS longitude_S			
	Unit	arcminute north of equator				arcminute east of Greenwich Meridian			
	Transform	Divide by 10000				Divide by 10000			
GPS Course, Speed and Altitude 681h 50Hz	Content	Course_U		GPS Speed_U		Altitude_S		NOT TRANSMITTED*	
	Unit	degree		m/s		m above sea level			
	Transform	Divide by 100		Divide by 100		none			
GPS Time and Date 682h 50Hz	Content	Day_U	Month_U	Year_U	Hour_U	Minute_U	Second_U	Thou of a second_U	
	Unit	DD	MM	YY	HH	MM	SS	TTTT	
	Transform	None	None	None	None	None	None	None	
Accelerom-eter 683h 100Hz	Content	Latitudinal acceleration_S		Longitudinal acceleration_S		Vertical acceleration_S		Accelerometer temperature_S	
	Unit	G		G		G		degC	
	Transform	Divide by 1000		Divide by 1000		Divide by 1000		Divide by 10	
Gyroscope 684h 100Hz	Content	Roll_S		Pitch_S		Yaw_S		Gyroscope temperature_S	
	Unit	degree/sec		degree/sec		degree/sec		degC	
	Transform	Divide by 10		Divide by 10		Divide by 10		Divide by 10	
GPS Status Information 685h 50Hz	Content	Horizontal dilution of precision		Fix quality indicator	Number of satellites	GPS mode letter	GPS status letter	NOT TRANSMITTED*	
	Unit	m		-	-	ASCII	ASCII		
	Transform	Divide by 10		None	None	None	None		

*These frames are only 6 bytes long. Not transmitted bytes are NOT the same as sending null values. Receive settings must accommodate the shorter frame.
The GPS-A10b does not send frame 684h. GPS frequency is limited to 10Hz.

6.2 CAN Configuration

The GPS-AG50 has configurable settings, allowing different CAN addresses amongst other options. To change these settings, commands frames are sent over CAN.

GPS commands are sent on the 7th frame from the base ID. With the default base ID at 680h, the command frame is 686h. To change the base ID, 3 messages must be sent. The first message is an unlock code that allows commands, the second is the command itself (an acknowledgment will be received) and the third will store the new settings. Note that this will also change the command frame to the 7th from the new ID.

Unlock Code:

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
1Eh	A4h	B0h	3Ch	B0h	D4h	11h	89h

Set CAN Base ID:

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5
Message index	GPS_M_SET_CONFIG (15h)	GPS_M_CONFIG_ITEM_CAN_BASE_ID (05h)	Base ID for CAN messages	

Response if Successful:

Byte 1	Byte 2	Byte 3
Message index	GPS_M_SET_CONFIG | 80h (95h)	GPS_R_OK (00h)

Store New Settings:

Byte 1	Byte 2	Byte 3
Message index	GPS_M_SET_CONFIG (15h)	GPS_M_CONFIG_ITEM_STORE (00h)

The Message Index is user definable to give the frame a recognisable ID. The response frame will mimic this ID.
Frame lengths must be exact. Not transmitted bytes are NOT the same as sending null values.

7 Document Revision History

2018-08-24	MH V1.0	Initial public release
2019-10-01	MH V1.1	Updated for PDU datastream V2