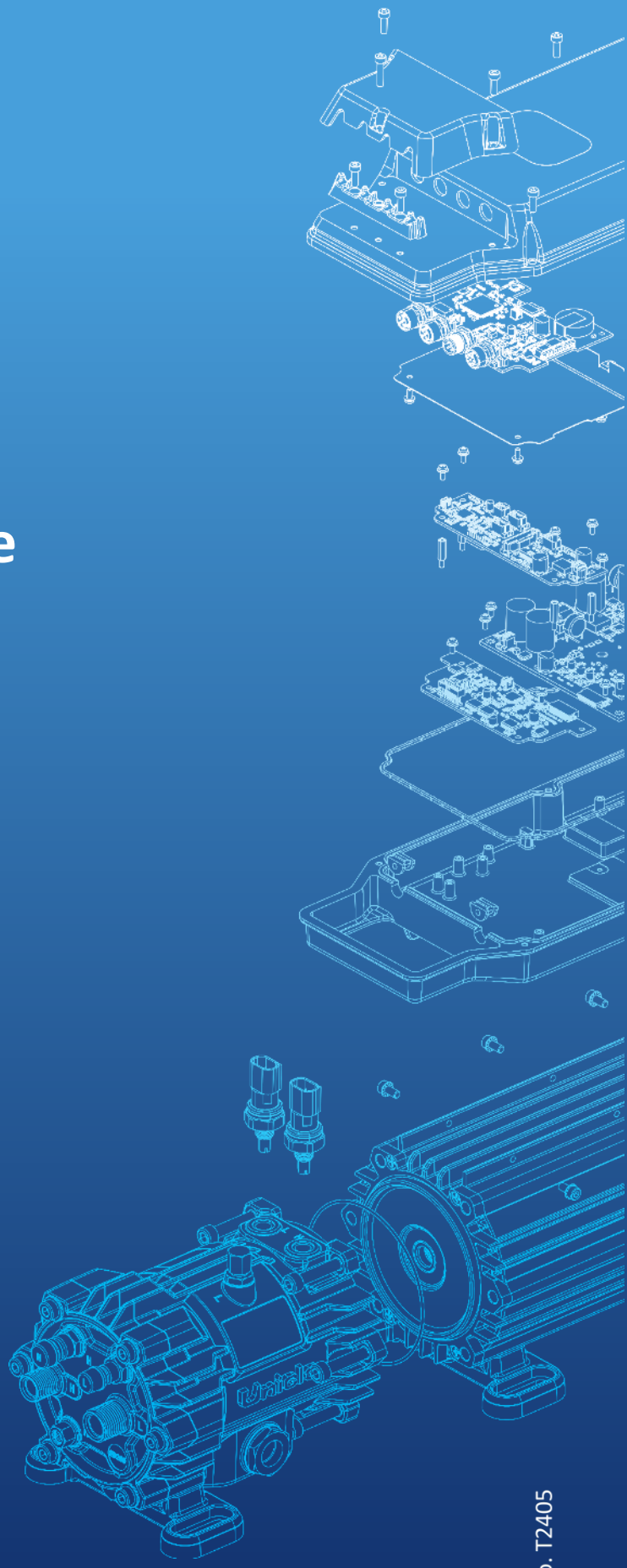
Unicla®

eDrive RS485 Interface

V16
2025-12-01



Unicla Cat.No. T2405

Table of Contents

Target release	Q4, 2024
Epic	TBA
Document owner	Ritesh Lal
Designer	Ritesh Lal
Tech lead	Mark Mitchell
Technical writers	TBA
QA	

Objective

Define implementation specifics of eDrive RS485 Interface.

Success metrics

Goal	Metric
Data Read	Status, Build Information, Configuration readable from eDrive
Configurable	RS485 implementation configurable by eConnect, eConfig etc.
Compressor Control	User selectable RS485 bus based control - Start, Stop, Speed Change and Ack Faults

Assumptions/Caveats

The following assumptions/caveats apply:

- RS485 implementation is half-duplex, hence all transfers are required to be master initiated (eDrive(s) are slaves). ***Note:*** *multiple eDrives can be connected to the same RS485 network by use of different device ids.*
- Configuration parameters settable by eConfig, eConnect and eMonitor
- Exposed RS485 data (as visible to a third party controller) available using a new application or expanded version of eCANViewer.
- Data on RS485 is **not** encrypted.

Strategy

Background

RS485 implementation using eDrive CAN implementation as the basis. RS485 capability will match (where possible) functionality and data exposed by eDrive CAN interface - eDrive CAN (Basic)

Implementation Specifics

All RS485 communication is initiated by the master. The following commands are possible:

Type	Payloads
Read	Status
Write	Command

Status

This includes critical data, refrigeration, some electrical values and safety flags fields:

Name	Type	Units	Payload								
Pressure	array (float)	bar-a	8	Suction				Discharge			
Temperature	array (float)	°C	8	Suction				Discharge			
Superheat	array (float)	K	8	Suction				Discharge			
Compressor/Motor	array (uint16)	…	8	RPM		Voltage (V)		Power (W)		Torque (Nm)	
Status	array	…	8	Status*		Control Mode*	% Demand	Remote Enable		Anti Short	X
Motor Temperatures	array (float)	°C	8	Motor				Switch			
Safety Flags (1 of 2)	array		8	Compressor Safety Flags - Run		Compressor Safety Flags- Start		Motor Safety Flags - Run		Motor Safety Flags - Start	
Safety Flags (2 of 2)	array		8	Electrical Safety Flags			X	X	X	X	X

* See below for interpretation, Status = *ED_OP_STATE* and Control Mode = *ED_CONTROL_MODE*

See *Interpreting Safety Flags* section below

X denotes reserved bytes

Build Information

Name	Type	Units	Payload								
Model (1 of 2)	ASCII	-	8								
Model (2 of 2)	ASCII	-	8								
Serial Number (1 of 2)	ASCII	-	8								
Serial Number (2 of 2)	ASCII	-	8								
Compressor Build	array	-	8	Refrigerant*	Oil*	Reserved		Min Speed (RPM)		Max Speed (RPM	
Electrical Build	array (uint)	...	8	Operating Voltage (V)		Power Rating (W)		Reserved			
Device Name (1 of 4)	ASCII	-	8								
Device Name (2 of 4)	ASCII	-	8								
Device Name (3 of 4)	ASCII	-	8								
Device Name (4 of 4)	ASCII	-	8								

* See below for interpretation: Refrigerant = *ED_REFRIGERANTS*. Oil = *ED_OILS*

Running Limits/ RS485 Config

Name	Type	Units	Payload								
Speed Control	array	RPM	8	Min Speed				Max Speed			
Pressure - Suction	array (float)	bar-a	8	Low Limit				High Limit			
Pressure - Discharge	array (float)	bar-a	8	Low Limit				High Limit			
Temperature - Suction	array (float)	°C	8	Low Limit				High Limit			
Temperature - Discharge	array (float)	°C	8	Low Limit				High Limit			
Superheat - Suction	array (float)	K	8	Low Limit				High Limit			
Superheat - Discharge	array (float)	K	8	Low Limit				High Limit			
RS485 Config	array		8	Device ID		Reserved		Reserved		Control Enabled	

Commands

Name	ID	Payload		
Read	0xC0	0		
Write	0xC1			
Set Run State		2	Speed Value - unsigned 16bit value	
Acknowledge Faults		0		

Set Run State requires a 16-bit speed value which needs to be between the configured min and max reported by eDrive. **Sending a value of 0 is treated as a stop command.**

Configurable Parameters

The following parameters are settable in the XC

Group Name	Parameter	Type/Unit	Default Value	Comments
XC Configuration	RS485 Enabled	bool	false	Control the availability of the RS485 module
XC RS485 Configuration	Mode	enum	Standard	Options are:
	Device ID	uint8	0x01	**0x00 is reserved for master device id**
	Baud Rate	enum	19200	Options are:

User Design Information

Message IDs

The following message IDs are present in the RS485 module

Name	ID	Read/Write
Status	0x01	Read Only
Build Information	0x02	
Running Limits	0x03	
Set Run State	0xC0	Write Only
Acknowledge Faults	0xC1	

Interpretation of consolidated data

Some data e.g. status is reported as a number. These are interpreted as follows:

```
ED_REFRIGERANTS = ['Unknown', 'R134A', 'R1234YF', 'R404A', 'R452A', 'R513A']
ED_OILS         = ['Unknown', 'PAG46', 'PAG46HD', 'PAG56', 'POE32', 'POE68']
ED_CONTROL_MODE = ['eConnect', 'Remote Enable', 'Variable', 'Variable + Remote Enable',
'External Serial']
ED_OP_STATE     = ['Not Ready', 'Starting', 'Running', 'Stopped', 'Wait - Compressor
Safety', 'Wait - Anti Short', 'Fault - Compressor Safety', 'Fault - Motor', 'Fault -
FMC', 'Fault - Electrical', 'Fault - Configuration', 'Fault - XC Comms', 'Locked',
'Motor Thermal Inhibit', 'Oil Balance Running', 'Oil Balance Required', 'Oil Balance
Run Wait']
```

Where the first entry is equal to 0. So in the case of Refrigerant, value = 2 means R123YF.

Interpreting Safety Flags

Safety Flags are presented in a packed format to minimise CAN traffic. These are published periodically together with other data. Each bit, if used, is set or unset depending on the existence of a condition. The flags are as follows:

Compressor Safety Flags - Run, Start

Bit	Flag
0	X
1	X
2	X
3	X
4	Low Side Superheat High
5	Low Side Superheat Low
6	High Side Superheat High
7	High Side Superheat Low
8	Low Side Temperature High
9	Low Side Temperature Low
10	Low Side Pressure High
11	Low Side Pressure Low
12	High Side Temperature High
13	High Side Temperature Low
14	High Side Pressure High
15	High Side Pressure Low

Motor Safety Flags - Run, Start

Bit	Flag
0	X
1	Switch Temperature Low
2	Switch Temperature High
3	Motor Temperature Low
4	Motor Temperature High
5	Motor Power Run Limit
6	Motor Power Absolute Limit
7	ST Motor Control Fault
8	ST Motor Control - High FOC Rate
9	ST Motor Control - Over voltage
10	ST Motor Control - Under Voltage
11	ST Motor Control - Over Temperature
12	ST Motor Control - Startup Fail
13	ST Motor Control - Speed Feedback Error
14	ST Motor Control - Over Current
15	ST Motor Control - Software Error

Electrical Safety Flags

Bit	Flag
0	FMC Data Invalid
1	FMC 3.3V Fault
2	FMC 5.0V Fault
3	FMC VMON 24V Low
4	FMC VMON 24V High
5	FMC VMON UDXC 24V Low
6	FMC VMON UDXC 24V High
7	FMC VMON UDXC 5V Low
8	FMC VMON UDXC 5V High
9	FMC CT 5V Low
10	FMC CT 5V High
11	X
12	X
13	X
14	X
15	X
16	CMC 3V3 Fault
17	CMC 5V0 Fault
18	CMC VMON 5V Low
19	CMC VMON 5V High
20	CMC AUX VMON VCC Low
21	CMC AUX VMON VCC High
22	CMC UDPC VCC Low
23	CMC UDPC VCC High

RS485 Communication Protocol

The RS485 protocol is loosely based on PPP frame structure and is defined in [RS485 Communication Protocol]

Communications Settings

Baud Rate	As configured by eConnect
Data Bits	8
Stop Bits	1
Parity	0
Flow Control	None

Message Structure

The RS485 Communication protocol is comprised of the following elements:

	Start of Frame (SOF)	**Header**								**Payload**	**CRC**	**End of Frame (EOF)**
Length	1	8								?	2	1
Value /	0x7E	Source	Destination	Command	Message ID	Sequence Number	Reserved	Reserved	Payload Size			0x7E

Start and End of Frame

The SOF and EOF, value 0x7E, is reserved and must **not** occur anywhere else in the message. This means data needs to be escaped - see below.

Header

The header contains the following message related items:

Device Address - Source/Destination

Defines the source and destination of the message

> Master device id is always 0x00

> In the case of global address, 0xFF, all devices will reply (if needed) with their individual device ids in the header

Command

This is either a read or write command. Successful read commands always get a reply in the way of the actual data. Write commands are not acknowledged. Write commands are limited and success/fail can be determined by other means.

Sequence Number

Inserted by master for its internal synchronisation purposes. The slave reply will contain this sequence number. See example below.

Reserved

Not used but must be set to 0. For future expansion.

Payload Size

The number of data bytes after header. Can be 0.

Cyclic Redundancy Check - CRC

A CRC is used to check correct reception of data. A lookup table is used:

```
crc16tab[] =
{

            0x0000, 0x1021, 0x2042, 0x3063, 0x4084, 0x50a5, 0x60c6, 0x70e7,
            0x8108, 0x9129, 0xa14a, 0xb16b, 0xc18c, 0xd1ad, 0xe1ce, 0xf1ef,
            0x1231, 0x0210, 0x3273, 0x2252, 0x52b5, 0x4294, 0x72f7, 0x62d6,
            0x9339, 0x8318, 0xb37b, 0xa35a, 0xd3bd, 0xc39c, 0xf3ff, 0xe3de,
            0x2462, 0x3443, 0x0420, 0x1401, 0x64e6, 0x74c7, 0x44a4, 0x5485,
            0xa56a, 0xb54b, 0x8528, 0x9509, 0xe5ee, 0xf5cf, 0xc5ac, 0xd58d,
            0x3653, 0x2672, 0x1611, 0x0630, 0x76d7, 0x66f6, 0x5695, 0x46b4,
            0xb75b, 0xa77a, 0x9719, 0x8738, 0xf7df, 0xe7fe, 0xd79d, 0xc7bc,
            0x48c4, 0x58e5, 0x6886, 0x78a7, 0x0840, 0x1861, 0x2802, 0x3823,
            0xc9cc, 0xd9ed, 0xe98e, 0xf9af, 0x8948, 0x9969, 0xa90a, 0xb92b,
            0x5af5, 0x4ad4, 0x7ab7, 0x6a96, 0x1a71, 0x0a50, 0x3a33, 0x2a12,
            0xdbfd, 0xcbdc, 0xfbbf, 0xeb9e, 0x9b79, 0x8b58, 0xbb3b, 0xab1a,
            0x6ca6, 0x7c87, 0x4ce4, 0x5cc5, 0x2c22, 0x3c03, 0x0c60, 0x1c41,
            0xedae, 0xfd8f, 0xcdec, 0xddcd, 0xad2a, 0xbd0b, 0x8d68, 0x9d49,
            0x7e97, 0x6eb6, 0x5ed5, 0x4ef4, 0x3e13, 0x2e32, 0x1e51, 0x0e70,
            0xff9f, 0xefbe, 0xdfdd, 0xcffc, 0xbf1b, 0xaf3a, 0x9f59, 0x8f78,
            0x9188, 0x81a9, 0xb1ca, 0xa1eb, 0xd10c, 0xc12d, 0xf14e, 0xe16f,
            0x1080, 0x00a1, 0x30c2, 0x20e3, 0x5004, 0x4025, 0x7046, 0x6067,
            0x83b9, 0x9398, 0xa3fb, 0xb3da, 0xc33d, 0xd31c, 0xe37f, 0xf35e,
            0x02b1, 0x1290, 0x22f3, 0x32d2, 0x4235, 0x5214, 0x6277, 0x7256,
            0xb5ea, 0xa5cb, 0x95a8, 0x8589, 0xf56e, 0xe54f, 0xd52c, 0xc50d,
            0x34e2, 0x24c3, 0x14a0, 0x0481, 0x7466, 0x6447, 0x5424, 0x4405,
            0xa7db, 0xb7fa, 0x8799, 0x97b8, 0xe75f, 0xf77e, 0xc71d, 0xd73c,
            0x26d3, 0x36f2, 0x0691, 0x16b0, 0x6657, 0x7676, 0x4615, 0x5634,
            0xd94c, 0xc96d, 0xf90e, 0xe92f, 0x99c8, 0x89e9, 0xb98a, 0xa9ab,
            0x5844, 0x4865, 0x7806, 0x6827, 0x18c0, 0x08e1, 0x3882, 0x28a3,
            0xcb7d, 0xdb5c, 0xeb3f, 0xfb1e, 0x8bf9, 0x9bd8, 0xabbb, 0xbb9a,
            0x4a75, 0x5a54, 0x6a37, 0x7a16, 0x0af1, 0x1ad0, 0x2ab3, 0x3a92,
            0xfd2e, 0xed0f, 0xdd6c, 0xcd4d, 0xbdaa, 0xad8b, 0x9de8, 0x8dc9,
            0x7c26, 0x6c07, 0x5c64, 0x4c45, 0x3ca2, 0x2c83, 0x1ce0, 0x0cc1,
            0xef1f, 0xff3e, 0xcf5d, 0xdf7c, 0xaf9b, 0xbfba, 0x8fd9, 0x9ff8,
            0x6e17, 0x7e36, 0x4e55, 0x5e74, 0x2e93, 0x3eb2, 0x0ed1, 0x1ef0
};
```

with a calculation function as below:

```
uint16_t calc_crc16(const char* const buffer, const uint16_t length)
  {
      uint16_t  crc = 0;
      uint16_t i;

      for (i = 0; i < length; i++)
      {
          crc = (crc << 8) ^ crc16tab[((crc >> 8) ^ buffer[i]) & 0x00FF];
      }
      return crc & 0xFFFF;
  }
```

CRC calculation is done before byte stuffing

CRC Example for Status Read Command with:

- Master address is 0x0
- eDrive address is 0x01
- Sequence number is 0x36

SOF and EOF are **not** used in CRC calculation

Command: 00 01 c0 01 36 00 00 00 **a7 70**, with 0xA7 & 0x70 being the CRC.

Byte Stuffing/ Unstuffing

Byte stuffing is used when sending data, and unstuffing is used on received data. This is to remove occurrences of SOF or EOF in the message. An escape byte, value 0x7D, is used to indicate where byte stuffing has occurred. As a result, instances of both 0x7E and 0x7D, in the message data need to be escaped.

Case 1 - SOF/EOF in data

0x7E is replaced with escape byte, 0x7D. Another byte, which is an XOR of 0x7E with 0x20, is added to the data

Case 2 - Escape byte in data

0x7D is left as is. Another byte, which is an XOR of 0x7D with 0x20, is added to the data.

Byte unstuffing is the reverse process where the received data is checked for occurrences of escape byte, 0x7D. The actual data byte is then an XOR of the next byte with 0x20.

Examples

Status Read from Device Id 0x01

Master → Slave (Command)

SOF	Source	Destination	Command	Message ID	Sequence No	Reserved		Payload	Data	CRC		EOF
0x7E	0x00	0x01	0xC0	0x01	0xXX	0x00	0x00	0x00	-	0xXX	0xXX	0x7E

Slave → Master (Reply)

SOF	Source	Destination	Command	Message ID	Sequence No	Reserved		Payload	Data	CRC		EOF
0x7E	0x01	0x00	0xC0	0x01	0xXX	0x00	0x00	0x40	64bytes	0xXX	0xXX	0x7E

Set Run State on Device Id 0x01

Master → Slave (Command Speed to 1000RPM)

SOF	Source	Destination	Command	Message ID	Sequence No	Reserved		Payload	Data		CRC		EOF
0x7E	0x00	0x01	0xC1	0xC0	0xXX	0x00	0x00	0x02	0x03	0xE8	0xXX	0xXX	0x7E

Unicla®

Unicla International Limited
Hong Kong: +852 2422 0180
Australia: +61 7 5549 4000
www.unicla.hk
sales@unicla.hk