The Extended CAN Message 29-bit Identifier is structured as follows:

Module Address (2-bit)	Module ID (4-bit)	Data Header (3-bit)	Message Type (4-bit)	Data ID (16-bit)

- **Module Address (2-bit):** These bits represent the specific address of the module within the network. With 2 bits, there are 4 possible addresses (00, 01, 10, 11). So, a maximum of 4 modules of the same type can be connected.
- **Module ID (4-bit):** These bits identify the type of module. With 4 bits, there are 16 possible module IDs (0000 to 1111).
- **Data Header (3-bit):** These bits provide additional information about the data being sent. With 3 bits, there are 8 possible headers (000 to 111).
- **Message Type (4-bit):** These bits specify the type of message being sent. With 4 bits, there are 16 possible message types (0000 to 1111).
- **Data ID (16-bit):** These bits represent the specific data id or payload associated with the message. With 16 bits, there are 65536 possible data IDs (0000000000000000 to 1111111111111111).

Types of Modules	Module Address	Module ID
ETU	0	0
TCP	0	1
TM1	0	2
TM2	1	2
DIO1	0	3
DIO2	1	3
DIO3	2	3
DIO4	3	3
EL	0	4
BLE	0	5

General

Data Header (3-bit)		**Message Type (4-bit)**		**Data ID (16-bit)**	**Frame Type**	**DLC**	**Process Description**
2	Breaker Data Status	0	Data Read Request	UID	REMOTE	-	Breaker Status Read: The initiator module sends 'Data Read Request' with 'UID' of the first parameter. The data owner Module sends 'Data Read Response' with the data bytes of the queried parameters. Please note that the Data Read Request is sent only once, Data Read Response and Data Read Acknowledgement are the frames needed to be communicated from the initiator and target modules respectively for sending larger payload data. Example for the same is explained below.
		1	Data Read Response	Same as Request	DATA	8	
		9	Data Read Acknowledgement	Same as Request	DATA	8	
7	Heartbeat	14 (E)	Data Transmit	UID	DATA	8	This is a process defined so that a Module can transmit uptime and health data without any request using 'Heartbeat' message.

Example 1: In case of TCP to ETU Communication over CAN – Querying for retrieving Breaker Data Status:

a. The **Module Address** of TCP Module is 0x00.
b. The initiator module is TCP Module having **Module ID** 0x01.
c. The **Data Header** is Breaker Data Status 0x02.
d. The **Message Type** is **Data Read Request** i.e. 0x00.
e. The UID i.e. the **Data ID** is Breaker Data Status' Start Address i.e. 0x0333.

The final CAN Extended ID is generated as **0x01200333.** **(Query from TCP to ETU)**

Example 1: In case of ETU to TCP Communication over CAN – Sending response to the above Query for Breaker Data Status:

a. The **Module Address, Module ID, Data Header** and **Data ID** will remain the same as Query.
b. The **Message Type** is **Data Read Response** i.e. 0x01.

The final CAN Extended ID is updated to **0x01210333.** **(Response from ETU to TCP)**

An elaborate example is shown below with Message ID and DLC Data. Please refer to Section (B).

General

CAN v2.0B Messages for TCP Modules communicating with ETU Module.

A. Heartbeat Message (Initiator Node/Module - Ethernet):

The "Heartbeat" Message refers to a periodic message broadcast by each type of module on the CAN bus. This message serves as a health check, ensuring that the node is still active and functioning correctly.

The CAN Extended Message ID is **0x017E0333** and the CAN Frame Format is **Classic CAN** and DLC is **8 bytes**.

The DLC Payload Structure’s first 4 bytes consist of the uint32_t upTimeTCP and remaining three bytes are 0:

```
CAN_HeartBeatMessage.ui8_Data[0] = (uint8_t)(upTimeTCP >> 24);
CAN_HeartBeatMessage.ui8_Data[1] = (uint8_t)(upTimeTCP >> 16);
CAN_HeartBeatMessage.ui8_Data[2] = (uint8_t)(upTimeTCP >> 8);
CAN_HeartBeatMessage.ui8_Data[4] = (uint8_t)upTimeTCP;
CAN_HeartBeatMessage.ui8_Data[4] = 0;
CAN_HeartBeatMessage.ui8_Data[5] = 0;
CAN_HeartBeatMessage.ui8_Data[6] = 0;
CAN_HeartBeatMessage.ui8_Data[7] = 0;
```

CAN Frame as seen on the bus:

```
TCP Heartbeat:    0x017E0333 | 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x00 | Classic CAN | 8 bytes | RTR Data
TCP Heartbeat:    0x017E0333 | 0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x00 | Classic CAN | 8 bytes | RTR Data
TCP Heartbeat:    0x017E0333 | 0x00 0x00 0x00 0x03 0x00 0x00 0x00 0x00 | Classic CAN | 8 bytes | RTR Data
```

This CAN_HeartBeatMessage is to be transmitted every 500ms on the CAN Bus without expecting any acknowledgement from any other nodes/modules.

B. Data Read Request Message (Initiator Node/Module – Ethernet & Target Node/Module – ETU):

This is a message request from the TCP Module to the ETU Module to receive a struct data.

We have a struct data which is to be transmitted over CAN.

General

```
typedef struct __attribute__((packed))
{
    uint16_t data1;
    uint16_t data2;
    uint16_t data3;
    uint16_t data4;
    uint16_t data5;
    uint16_t data6;
    uint16_t data7;
    uint16_t data8;
    uint16_t data9;
    uint16_t data10;
    uint16_t data11;
    uint16_t data12;
    uint16_t data13;
    uint16_t data14;
    uint16_t data15;
    uint16_t data16;
    uint16_t data17;
    uint16_t data18;
    uint16_t data19;
    uint16_t data20;
    uint16_t data21;
    uint16_t data22;
    uint16_t data23;
    uint32_t data24;
    uint32_t data25;
    uint16_t data26;
} BR_DATA;
```

Converting the above typedef struct to a uint8_t buffer_BR_DATA **[]**

```
uint8_t buffer_BR_DATA[] =
{
    0x01, 0x00,               // data1
    0x00, 0x00,               // data2
    0x01, 0x00,               // data3
    0x02, 0x00,               // data4
    0x01, 0x00,               // data5
    0x00, 0x00,               // data6
    0x01, 0x00,               // data7
    0x96, 0x00,               // data8
    0x05, 0x00,               // data9
    0x91, 0x00,               // data10
    0x0A, 0x00,                // data11
    0x03, 0x00,                // data12
    0x01, 0x00,                // data13
    0x02, 0x00,                // data14
    0x04, 0x00,                // data15
    0x01, 0x00,                // data16
    0x32, 0x00,                // data17
    0x30, 0x00,                // data18
    0x0A, 0x00,                // data19
    0x05, 0x00,                // data20
    0x04, 0x00,                // data21
    0x06, 0x00,                // data22
    0x03, 0x00,                // data23
    0x10, 0x27, 0x00, 0x00,    // data24
    0xF4, 0x01, 0x00, 0x00,    // data25
    0x01, 0x01                 // data26
};
```

CAN DLC and Extended Message ID Explanation:

1. Calculate the number of CAN Frames:

Number of CAN Frames = (sizeof(buffer_BR_DATA) / 5).

Important: If there is a remainder, round up to the next whole number.

2. Querying and Reception of CAN Frames:

a. An RTR frame of Remote type is sent by the **TCP Module** using Message ID: **0x01200333** to which the **ETU Module** will send a RTR Data Frame as a response with Message ID: **0x01210333** appending CAN Frame Number (1 byte), then first 5 bytes of the buffer_BR_DATA[] (5 bytes) to the DLC. Lastly calculating the 16-bit CRC of bytes 0 to 5 and appending the same (2 bytes) to the DLC as shown below:

```
            1 byte -----------------------------------  5 bytes  --------------------------------  2bytes
CAN Frame Number (DLC byte 0)              Payload Data (DLC bytes 1 to 5)          16-bit CRC (DLC bytes 6 and 7)
```

```
TCP Request:      0x01200333 | --------------------------------------- | Classic CAN | 0 bytes | RTR Remote

ETU Response 1:   0x01210333 | 0x01 0x01 0x00 0x00 0x00 0x01 0xFF 0xFC | Classic CAN | 8 bytes | RTR Data
```

b. The **TCP Module** checks the CRC and the CAN Frame Number, if the CRC check is successful then the **TCP Module** sends an Acknowledgement CAN Frame of Message ID: **0x01490333** clearing the Payload Data (i.e., settings the bytes as 0) but retaining the CAN Frame Number. The CRC of bytes 0 to 5 is then recalculated and appended to the DLC of the new CAN Frame.

```
TCP Frame Ack 1:  0x01290333 | 0x01 0x00 0x00 0x00 0x00 0x00 0xFF 0xFE | Classic CAN | 8 bytes | RTR Data
```

c. **ETU Module** receives the above Acknowledgement CAN Frame.

- The DLC of the new ETU response frame updates here. In the DLC; byte 0 CAN Frame Number increments by 1; followed by this, the next 5 bytes of buffer_BR_DATA[] are appended.
- Lastly calculating the 16-bit CRC of bytes 0 to 5 and appending the same (2 bytes) to the DLC.
- The Message ID is updated by adding payload data size (5 bytes) to the previous response sent by ETU i.e., **0x01210333 + 0x05** = **0x01210338**.

```
ETU Response 2:   0x01210338 | 0x02 0x00 0x02 0x00 0x01 0x00 0xFF 0xFA | Classic CAN | 8 bytes | RTR Data
```

- The TCP Module then replies with the Acknowledgement upon successful reception of data.

```
TCP Frame Ack 2:  0x01290338 | 0x02 0x00 0x00 0x00 0x00 0x00 0xFF 0xFD | Classic CAN | 8 bytes | RTR Data
```

d. Steps (a) to (d) are repeated for all transmitting all the data frames.

```
ETU Response 3:   0x0121033D | 0x03 0x00 0x00 0x01 0x00 0x96 0xFF 0x65 | Classic CAN | 8 bytes | RTR Data
```

```
TCP Frame Ack 3:  0x0129033D | 0x03 0x00 0x00 0x00 0x00 0x00 0xFF 0xFC | Classic CAN | 8 bytes | RTR Data
ETU Response 4:   0x01210342 | 0x04 0x00 0x05 0x00 0x91 0x00 0xFF 0x65 | Classic CAN | 8 bytes | RTR Data
TCP Frame Ack 4:  0x01290342 | 0x04 0x00 0x00 0x00 0x00 0x00 0xFF 0xFB | Classic CAN | 8 bytes | RTR Data
ETU Response 5:   0x01210347 | 0x05 0x0A 0x00 0x03 0x00 0x01 0xFF 0xEC | Classic CAN | 8 bytes | RTR Data
TCP Frame Ack 5:  0x01290347 | 0x05 0x00 0x00 0x00 0x00 0x00 0xFF 0xFA | Classic CAN | 8 bytes | RTR Data
ETU Response 6:   0x0121034C | 0x06 0x00 0x02 0x00 0x04 0x00 0xFF 0xF3 | Classic CAN | 8 bytes | RTR Data
TCP Frame Ack 6:  0x0129034C | 0x06 0x00 0x00 0x00 0x00 0x00 0xFF 0xF9 | Classic CAN | 8 bytes | RTR Data
ETU Response 7:   0x01210351 | 0x07 0x01 0x00 0x32 0x00 0x30 0xFF 0x95 | Classic CAN | 8 bytes | RTR Data
TCP Frame Ack 7:  0x01290351 | 0x07 0x00 0x00 0x00 0x00 0x00 0xFF 0xF8 | Classic CAN | 8 bytes | RTR Data
ETU Response 8:   0x01210356 | 0x08 0x00 0x0A 0x00 0x05 0x00 0xFF 0xE8 | Classic CAN | 8 bytes | RTR Data
TCP Frame Ack 8:  0x01290356 | 0x08 0x00 0x00 0x00 0x00 0x00 0xFF 0xF7 | Classic CAN | 8 bytes | RTR Data
ETU Response 9:   0x0121035B | 0x09 0x04 0x00 0x06 0x00 0x03 0xFF 0xE9 | Classic CAN | 8 bytes | RTR Data
TCP Frame Ack 9:  0x0129035B | 0x09 0x00 0x00 0x00 0x00 0x00 0xFF 0xF6 | Classic CAN | 8 bytes | RTR Data
ETU Response 10:  0x01210360 | 0x0A 0x00 0x10 0x27 0x00 0x00 0xFF 0xBE | Classic CAN | 8 bytes | RTR Data
TCP Frame Ack 10: 0x01290360 | 0x0A 0x00 0x00 0x00 0x00 0x00 0xFF 0xF5 | Classic CAN | 8 bytes | RTR Data
ETU Response 11:  0x01210365 | 0x0B 0xF4 0x01 0x00 0x00 0x01 0xFE 0xFE | Classic CAN | 8 bytes | RTR Data
TCP Frame Ack 11: 0x01290365 | 0x0B 0x00 0x00 0x00 0x00 0x00 0xFF 0xF4 | Classic CAN | 8 bytes | RTR Data
ETU Response 12:  0x0121036A | 0x0C 0x01 0x00 0x00 0x00 0x00 0xFF 0xF2 | Classic CAN | 8 bytes | RTR Data
TCP Frame Ack 12: 0x0129036A | 0x0C 0x00 0x00 0x00 0x00 0x00 0xFF 0xF3 | Classic CAN | 8 bytes | RTR Data
```

General