UGRacing CAN Packet Specification

This document specifies eleven different formats of CAN packets that can be sent over the CAN bus in the UGRacing car. This specification is provided to simplify the process of developing data loggers, displays and pit software which use the data sent by subsystems within the car.

In addition to this, the GUFST pit software is capable of sending data over the bus in any of these formats to allow for configuration of devices on the bus.

Simple Integer Formats

1. 8x Byte Format

Use this format when sending 8 individual bytes of data. Any unused bytes should be set to zero. The bytes can be signed (-128 to 127) or unsigned (0 to 255).

BYTE0	BYTE1	BYTE2	BYTE3	BYTE4	BYTE5	BYTE6	BYTE7
CANDSR0	CANDSR1	CANDSR2	CANDSR3	CANDSR4	CANDSR5	CANDSR6	CANDSR7

```
char BYTE0 = -20;
unsigned char BYTE1 = 0x0A;
CANDSR0 = BYTE0;
CANDSR1 = BYTE1;
CANDSR2 = 0; //etc
```

2. 4x Integer Format

Use this format to send signed (-32768 to 32767) or unsigned (0 to 65535) integer values. Any unused bytes should be set to zero.

INT0MSB	INT0LSB	INT1MSB	INT1LSB	INT2MSB	INT2LSB	INT3MSB	INT3LSB
CANDSR0	CANDSR1	CANDSR2	CANDSR3	CANDSR4	CANDSR5	CANDSR6	CANDSR7

```
int INT0 = -10000;
unsigned int INT1 = 200;
CANDSR0 = (INT0>>8);
CANDSR1 = (INT0 & 0xFF);
CANDSR2 = (INT1>>8);
CANDSR3 = (INT1 & 0xFF);
CANDSR4 = 0; //etc
```

3. 2x Long Format

Use this format to send large signed (-2147483648 to 2147483647) or unsigned (0 to 4294967295) integers. Any unused bytes should be set to zero.

LONG0_MSB	LONG0_1	LONG0_2	LONG0_LSB	LONG1_MSB	LONG1_1	LONG1_2	LONG1_LSB
CANDSR0	CANDSR1	CANDSR2	CANDSR3	CANDSR4	CANDSR5	CANDSR6	CANDSR7

```
long LONG0 = 65764478;
CANDSR0 = (LONG0>>24) & 0xFF;
CANDSR1 = (LONG0>>16) & 0xFF;
CANDSR2 = (LONG0>>8) & 0xFF;
CANDSR3 = (LONG0 & 0xFF);
CANDSR4 = 0; //etc
```

Mixed Integer Formats

If different sizes of integer need to be sent, rather than using two CAN packets, the integer formats can be mixed using one of the following formats. Again, in any of these packets unused bytes should be set to zero.

4. 1 Long, 4 Bytes Format

LONG_MSB	LONG_1	LONG_2	LONG_LSB	BYTE0	BYTE1	BYTE2	BYTE3
CANDSR0	CANDSR1	CANDSR2	CANDSR3	CANDSR4	CANDSR5	CANDSR6	CANDSR7

5. 1 Long, 2 Integers Format

LONG_MSB	LONG_1	LONG_2	LONG_LSB	INT0MSB	INT0LSB	INT1MSB	INT1LSB
CANDSR0	CANDSR1	CANDSR2	CANDSR3	CANDSR4	CANDSR5	CANDSR6	CANDSR7

6. 2 Integers, 4 Bytes Format

INT0MSB	INT0LSB	INT1MSB	INT1LSB	BYTE0	BYTE1	BYTE2	BYTE3
CANDSR0	CANDSR1	CANDSR2	CANDSR3	CANDSR4	CANDSR5	CANDSR6	CANDSR7

7. 1 Long, 1 Integer, 2 Bytes Format

LONG_MSB	LONG_1	LONG_2	LONG_LSB	INT_MSB	INT_LSB	BYTE0	BYTE1
CANDSR0	CANDSR1	CANDSR2	CANDSR3	CANDSR4	CANDSR5	CANDSR6	CANDSR7

Decimal Formats

The following decimal formats have been specified to allow decimal values to be sent across the bus. The floating point values are converted to integers before transmission to avoid conflicts between the different floating point systems in use.

8. 2 Small Decimal Numbers

These packets allow for two numbers in the range (-32768 to 32767) to be sent across the bus, with an accuracy of up to 4 decimal places. Unused bytes should be set to zero.

INT0_MSB	INT0_LSB	FRACT0_MSB	FRACT0_LSB	INT1_MSB	INT1_LSB	FRACT1_MSB	FRACT1_LSB
CANDSR0	CANDSR1	CANDSR2	CANDSR3	CANDSR4	CANDSR5	CANDSR6	CANDSR7

From this data, decimal values are calculated as follows:

DECIMAL0 = INT0 + (FRACT0/10000)

The following C code stores numbers in this format:

```
double val0 = 2.5476;
double val1 = -5676.213454; //Value will be truncated to 4 decimal places
int int0 = (int)val0;
int fract0 = (int)((val0 - ((int)val0)) * 10000);
int int1 = (int)val1;
int fract1 = (int)((val1 - ((int)val1)) * 10000);
CANDSR0 = int0>>8;
CANDSR1 = int0 & 0xFF;
CANDSR2 = fract0 >>8;
CANDSR3 = fract0 & 0xFF;
//etc
```

9. 1 Large Decimal Number

These packets allow for the transmission of one large decimal number in the range (-2147483648 to 2147483647) with accuracy of 8 decimal places.

INT_MSB	INT_1	INT_2	INT_LSB	FRACT_MSB	FRACT_1	FRACT_2	FRACT_3
CANDSR0	CANDSR1	CANDSR2	CANDSR3	CANDSR4	CANDSR5	CANDSR6	CANDSR7

From this data, decimal values are calculated as follows:

DECIMAL = INT + (FRACT/100000000);

The following C code stores numbers in this format:

```
double value = 26567865.8786530;
long int_part = (long)value;
long fract_part = (long)((value - ((long)value)) * 100000000);
CANDSR0 = (int_part>>24) & 0xFF;
CANDSR1 = (int_part>>16) & 0xFF;
CANDSR2 = (int_part>>8) & 0xFF;
CANDSR3 = (int_part & 0xFF);
CANDSR4 = (fract_part>>24) & 0xFF;
CANDSR5 = (fract_part>>16) & 0xFF;
CANDSR6 = (fract_part>>8) & 0xFF;
CANDSR7 = (fract_part & 0xFF);
```

Text Formats

The following formats allow for text to be sent over the CAN bus.

10. 8 bytes of text

This format allows 8 bytes (8 characters) of text to be sent across the CAN bus. This is suitable for short text messages. Unused bytes should be set to zero.

CHAR0	CHAR1	CHAR2	CHAR3	CHAR4	CHAR5	CHAR6	CHAR7

CANDSR0	CANDSR1	CANDSR2	CANDSR3	CANDSR4	CANDSR5	CANDSR6	CANDSR7

```
const char * text = “string”;
CANDSR0 = text[0];
CANDSR1 = text[1];
CANDSR2 = text[2];
CANDSR3 = text[3];
CANDSR4 = text[4];
CANDSR5 = 0; //etc
```

11. Multiple packets for more than 8 characters

The CAN packet size significantly restricts the amount of text that can be sent over the bus. To allow for larger packets of text to be sent over the bus the following format using ASCII control characters is available to allow large amounts of text to be split over CAN packets.

The first packet should contain the ASCII Start of Text character, followed by the first part of the string:

0x02 (STX)	CHAR0	CHAR1	CHAR2	CHAR3	CHAR4	CHAR5	CHAR6
CANDSR0	CANDSR1	CANDSR2	CANDSR3	CANDSR4	CANDSR5	CANDSR6	CANDSR7

Any intermediate packets should contain the ASCII new page character followed by text. This ensures that any systems that did not receive the start of the text will disregard intermediate packets until the end of the text is reached.

0x0C (NP)	CHAR7	CHAR8	CHAR9	CHAR10	CHAR11	CHAR12	CHAR13
CANDSR0	CANDSR1	CANDSR2	CANDSR3	CANDSR4	CANDSR5	CANDSR6	CANDSR7

The last packet should contain the ASCII End of Text character after the last character of the string, with any unused bytes set to zero.

0x0C (NP)	CHAR14	CHAR15	CHAR16	0x03 (ETX)	0	0	0
CANDSR0	CANDSR1	CANDSR2	CANDSR3	CANDSR4	CANDSR5	CANDSR6	CANDSR7

The following C code example sends a 12 character string using two CAN packets:

```
const char * text = “abcdefghijkl”;
CANDSR0 = 0x02;         //Start of text
CANDSR1 = text[0];
CANDSR2 = text[1];
CANDSR3 = text[2];
CANDSR4 = text[3];
CANDSR5 = text[4];
CANDSR6 = text[5];
```

```
CANDSR7 = text[6];
Insert code to send CAN packet here
CANDSR0 = 0x0C;         //New Page
CANDSR1 = text[7];
CANDSR2 = text[8];
CANDSR3 = text[9];
CANDSR4 = text[10];
CANDSR5 = text[11];
CANDSR6 = 0x03;         //End of text
CANDSR7 = 0;
Insert code to send CAN packet here
```