MGL Avionics flight data interface specification

Preliminary

This is a design specification and subject to change

Document revision: 6

28 February 2012

General

This document describes the raw data format used for transferring flight data information from an EFIS system to an MGL avionics flight data recording device (black box flight recorder) or other devices.

The following message IDs are defined:

Message 01 – Primary flight data
Message 02 – GPS data
Message 03 – Attitude data
Message 04 – Various inputs (flaps, gear, etc)
Message 05 – Traffic file
Message 06 – Alert/Warning/Information text message
Message 10 – Engine message
Message 11 – Fuel tank levels
Message 30 – Navigation data (Active navigation information, HSI, VSI, GSI)
Message 31 – Secondary navigation data
Message 35 - COM
Message 41 – Flight plan header
Message 42 – Flight plan item
Message 200-255 – Vendor specific. Please obtain information from your vendor.

Data is transmitted over standard RS232 signaling using the TX line. RX line is not used by transmitting equipment.
Baudrate is set to 115200 at 8 bits per character, 1 start and 1 stop bit.
This allows transmission of up to 11520 bytes per second.

Message data is binary and arranged to suit processors that have word boundary access limitations (such as most ARM implementations). Messages are arranged to favor DMA message pipes common in modern micro controllers to allow message reception without microprocessor core involvement (Direct UART -> Memory transfers).
Receiving equipment must be able to handle a message length of up to 276 bytes total (up to 264 bytes data).
Messages can be securely received by means of a strong message start synchronization (05+02+Message length+Message length xor 0xFF). This prevents false synchronization on message content. Data content can be verified using a standard CRC 32 bit checksum giving a high level of data integrity confidence.
A message length of 0x00 should be interpreted as 0x100, I.e. 256 bytes.

Each message contains a byte that shows its average message rate per second. A further byte shows the message number within that second. This can be used at the receiving side to detect missing messages. This number counts up from 1 and is reset to 1 at the start of the next system second. For message rates of 1 per second, this byte alternates between 0 and 1 for every transmission.

It is acceptable for the EFIS to vary the rate per message on a per second bases. For example, one second it may send 10 messages of a particular type, the next second it may decide to send 5 messages. However, the EFIS MUST update the message rate information byte in the message. Message rate changes MUST only occur when the message counter is at its reset value (at start of the current second).

For flight data recording purposes, the data recorder may skip messages that exceed the set recording rate. For example, a message may be transmitted 10 times per second but the flight data recorder elects to only record one message per second.

Note: Message count is assumed to be asynchronous and would be reset to 1 at the start of a system second. It is permissible for the transmitting device to skip messages where this is needed for bandwidth control or it may be due to system load reasons. Due to the typical asynchronous nature of these transmissions with respect to time it is also possible for a message count to exceed the message rate number. For example, a message with a rate of 4 per second may have a message count field of 5 before being reset to 1 for the next second.

Devices should not use message count fields or message arrival times for internal timing as these are not guaranteed.

Data types

Longint	32 bit signed integer
Smallint	16 bit signed integer
Word	16 bit unsigned integer
Byte	8 bit unsigned integer
String	Byte based string of characters. The first byte is length of string and any characters in ASCII follow. Unused locations are to be treated as “don't care”. An empty string has a “0” in the first location. Example: String[6] – This string can have up to 6 characters and it occupies 7 bytes (length byte plus six characters). This is different to a “C” string but has the advantage that any value can be used as a character so this method can be used to store flexible length general data.

Byte order

Data types consisting of more than one byte are sent LSB first and MSB last

Unknown or invalid values

Any unknown or invalid quantity in a message should be set to zero unless otherwise mentioned in the specification of the message.

Message length byte in header

The message length byte refers to length of data portion (excluding header, checksum plus 8 bytes default message length).

The data portion of a message MUST be at least 9 bytes in size. This is the case for all standard messages. If the data portion is 9 bytes in size, the length byte has a value of “1” and the XOR byte following has a value of 0xFE (0x01 xor 0xFF).

This scheme allows the transmission of a 256 byte data page with a private 8 byte header.

Total, maximum number of bytes in a message is thus: 276 bytes

8 bytes header (starting at DLE)

+

9 to 264 bytes of data (maximum 256+8 bytes private header)

4 bytes checksum

If you parse a packet of data and you have just received the XOR'ed version of the length byte and thus can verify the length, you need to receive a further “length byte” + 16 bytes to receive all outstanding bytes in the packet including the 4 byte checksum.

Checksum location

Checksum location must be on a 4 byte boundary, counting from the DLE - filler bytes are inserted if needed to ensure this. This is to allow even 32 bit word access to the checksum in a receiver using an ARM processor or similar with word size boundary access restrictions or to utilize CRC32 hardware available in some processor chips that may be restricted to using word (32 bit) aligned data.

Message 01: Primary flight

This message should be sent by the EFIS at a recommended rate of 5 per second. Rates from 1 to 30 per second are acceptable based on the systems requirements.

DLE:	byte;	0x05	
STX:	byte;	0x02	
MessageLength:	byte;	0x18	36 bytes following MessageVersion - 12
MessageLengthXOR:	byte;	0xE7	
MessageType:	byte;	0x01	
MessageRate:	byte;	0x05	
MessageCount:	byte;	Message Count within current second	
MessageVersion:	byte;	0x01	
PAltitude:	longint;	Pressure altitude in feet	
BAltitude:	longint;	Pressure altitude in feet, baro corrected	
ASI:	word;	Indicated airspeed in 10^{th} Km/h	
TAS:	word;	True airspeed in 10^{th} Km/h	
AOA:	smallint;	Angle of attack in tenth of a degree	
VSI:	smallint;	Vertical speed in feet per minute	
Baro:	word;	Barometric pressure in 10^{th} millibars (actual measurement from altimeter sensor, actual pressure)	
Local:	word;	Local pressure setting in 10^{th} millibars (QNH)	
OAT:	smallint;	Outside air temperature in degrees C	
Humidity:	byte;	0-99%. If not available 0xFF	
SystemFlags:	Byte;	See description below	
Hour,Minute, Second,Date, Month,Year:	bytes;	Time as set in RTC. 24 hour format, two digit year.	
FTHour,FTMin:	bytes;	Flight time since take off. Hours, minutes.	
Checksum	longint;	CRC32	

SystemFlags:

bit 0	0: no flight active	1: flight active
bit 1	0: no OAT sensor	1: OAT sensor detected
bit 2	0: no humidity sensor	1: Humidity sensor detected

Message 02: GPS Message

This message contains basic GPS information. This message should be sent at a rate compatible with the systems GPS update rate (typically from 1 to 10 per second). Minimum rate is 1 per second.

DLE:	byte;	0x05	
STX:	byte;	0x02	
MessageLength:	byte;	0x24	48 bytes following MessageVersion - 12
MessageLengthXOR:	byte;	0xDB	
MessageType:	byte;	0x02	
MessageRate:	byte;	0x04	
MessageCount:	byte;	Message Count within current second	
MessageVersion:	byte;	0x01	
Latitude:	longint;	Latitude in degrees / 180.000 (+ = North)	
Longitude:	longint;	Longitude in degrees / 180.000 (+ = East)	
GPSAltitude:	longint;	Altitude from GPS in feet	
AGL:	longint;	Altitude above ground level as determined by terrain	
North velocity:	longint;	velocity towards north cm/s	
East velocity:	longint;	velocity towards east cm/s	
Down velocity:	longint;	velocity towards down cm/s	
GroundSpeed:	word;	Ground speed from GPS in 10^{th} Km/h	
TrackTrue:	word;	True track from GPS. 10^{th} of a degree	
Variation:	smallint;	Magnetic variation in 10^{th} of a degree. Negative is west.	
GPS	byte;	See description below	
SatsTracked:	byte;	Number of satellites tracked	
SatsVisible:	byte;	Total number of satellites visible	
HorizontalAccuracy:	byte;	Horizontal GPS accuracy estimate in feet	
VerticalAccuracy:	byte;	Vertical GPS accuracy estimate in feet	
GPS capability:	byte;	See below	
RAIM status:	byte;	See Raim information below	
RAIM HError:	byte;	Horizontal expected error	
RAIM VError:	byte;	Vertical expected error	
PaddingByte1:	byte;	0x00 For alignment	
Checksum	longint;	CRC32	

GPS byte

This byte shows the GPS mode:

0 : Acquiring
1: GPS internal dead reckoning
2: 2D fix
3: 3D fix
4: 2D fix EFIS dead reckoning (IMU)
5: 3D fix EFIS dead reckoning (IMU)

RAIM information

Status: 0: no satellite fail detected, else ID of most likely failed satellite
HError,VError: Horizontal and Vertical error in feet, based on using only satellites that passed the RAIM test.

Note: GPS data items are validated against this value. All GPS derived values are invalid if GPS byte is 0. GPS altitude is invalid if GPS byte is not 3 or 5.
The EFIS system may use dead reckoning to arrive at a higher position update rate than the GPS system can provide, for example using IMU. If the current data is based on a dead reckoning estimate, the GPS mode is 4 or 5.

GPS capability

Bit 0:	0: GPS not designed to DO-229	1: GPS designed to DO-229 Beta 1 or higher
Bit 1:	0: Not WAAS capable or disabled	1: WAAS capable and enabled
Bit 2:	0: No RAIM functionality	1: RAIM functional and enabled
Bit 3:	0: GPS can track less than 12 sats	1: GPS can track more than 11 sats
Bit 4:	0: GPS cannot use Glonast/Galileo	1: GPS can use Glonast/Galileo

Message 03: Attitude

This message is sent typically at the processed AHRS rate, not the native AHRS rate. Transmission rates are typically related to EFIS screen refresh or internal image drawing update rates. Typical rates are from 1 to 50 messages per second. Recommended rates would be from 10 to 25 to ensure smooth image creation where this is needed.

DLE:	byte;	0x05	
STX:	byte;	0x02	
MessageLength:	byte;	0x14	32 bytes following MessageVersion - 12
MessageLengthXOR:	byte;	0xEB	
MessageType:	byte;	0x03	
MessageRate:	byte;	0x0A	
MessageCount:	byte;	Message Count within current second	
MessageVersion:	byte;	0x01	
HeadingMag:	word;	Magnetic heading from compass. 10^{th} of a degree	
PitchAngle:	smallint;	AHRS pitch angle 10^{th} of a degree	
BankAngle:	smallint;	AHRS bank angle 10^{th} of a degree	
YawAngle:	smallint;	AHRS yaw angle 10^{th} of a degree (see notes below)	
TurnRate:	smallint;	Turn rate in 10^{th} of a degree per second	
Slip:	smallint;	Slip (ball position) -50 (left) to +50 (right)	
GForce:	smallint;	Acceleration acting on aircraft in Z axis (+ is down)	
LRForce:	smallint;	Acceleration acting on aircraft in left/right axis (+ if right)	
FRForce:	smallint;	Acceleration acting on aircraft in forward/rear axis (+ is forward)	
BankRate:	smallint;	Rate of bank angle change (See notes on units)	
PitchRate:	smallint;	Rate of pitch angle change	
YawRate:	smallint;	Rate of yaw angle change	
SensorFlags:	byte;	See description below	
PaddingByte1:	byte;	0x00 For alignment	
PaddingByte2:	byte;	0x00 For alignment	
PaddingByte3:	byte;	0x00 For alignment	
Checksum	longint;	CRC32	

SensorFlags:

bit 0	0: No magnetic compass	1: Magnetic compass detected
bit 1	0: No AHRS	1: AHRS detected
bit 2	0: No GPS	1: GPS detected and operational
bit 3	0: No meaning	1: AHRS over range or compromised
bit 4	0: AHRS is gyro	1: AHRS does not use gyros (GPS derived)
bit 5	0: No X/Y acceleration	1: X/Y acceleration is measured
bit 6	0: No rates	1: Rates are provided

Accelerations

Accelerations are in 100^{th} of a G.

Yaw Angle

Yaw angle is system specific. It may be referenced to North (true or magnetic) or it can be freely drifting, depending on the underlying hardware implementation.

Gyro Rates

Rates are transmitted in a 16 bit signed format involving two scaling factors chosen depending on the rate at the time.
For sensor rates less than 150 degrees per second:
Value is in 100^{th} of a degree per second. Highest value is thus +14999 or -14999
For sensor rates higher or equal to 150 degrees per second:
Value is in 10^{th} of a degree per second +/- 1500 +/-15000 depending on direction.

Examples:

Rate is 89.45 degrees per second: Value is 8945.
Rate is 345.3 degrees per second: Value is 16953. (3453-1500+15000).

Positive numbers: Bank right, Pitch up, Yaw right.
Negative numbers: Bank left, Pitch down, Yaw left.

Euler angle ranges

Bank Angle range: -1800 to +1799 – positive is bank right
Pitch Angle range: -900 to +899 – positive is pitch up
Yaw Angle range: 0 to 3599

Message 04: Various input states and signals

This message is sent typically at a rate of 2 per second. It contains readings related to analog and digital inputs that can be populated as needed by a vendor.

```
DLE:                     byte;     0x05
STX:                     byte;     0x02
MessageLength:           byte;     Depends on message content
MessageLengthXOR:        byte;     Depends on message content
MessageType:             byte;     0x04
MessageRate:             byte;     0x02
MessageCount:            byte;     Message Count within current second
MessageVersion:          byte;     0x01
NumberOfAnalogInputs: byte;        Number of analog input reading in this message
NumberOfDigitalInputs: byte;       Number of digital input readings in this message
Gear 1 state:            byte;
Gear 2 state:            byte;
Gear 3 state:            byte;
Gear 4 state:            byte;
Gear 5 state:            byte;
FlapPosition:            byte;
FlapPositionAnalog:      smallint;
PitchTrimPosition:       smallint;
BankTrimPosition:        smallint;
YawTrimPosition:         smallint;
Digital:                 longint; //Up to 32 bits for digital input states
Analog:                  array[0..NumberOfAnalogInputs-1] of word; //See notes
Checksum                 longint;  CRC32
```

The number of analog inputs must be even (0 is allowed). This is to ensure long word (4 byte) alignment of checksum.
Gear states are up to the vendor to define. They can be used for anything and are not necessarily used for "landing gear".
For landing gear the states are typical:

0	Gear down
1-254	Gear in intermediate position (can be a scaled value of actual position if known)
255	Gear up

Analog values are 16 bit, usually raw ADC values but may be scaled to present quantities etc. For MGL systems the values are scaled based on settings on the EFIS.
Flap positions are:

0	Unknown
1	Flap up
2	Flap 1
3	Flap 2
4	Flap down
10	Flap up, negative (some aircraft allow negative flap positions for fast cruise)

Flap position analog – as decided by vendor
Trim positions – as decided by vendor. It is recommended to use 0 as "neutral"

Message 05: Traffic file

This message is sent at a rate of once per second. It contains up to 32 traffic items.
Traffic items may be sorted in threat order if the system supports this (this is identified by the traffic mode bit).
Traffic location is processed by the EFIS regardless of source and shown in latitude and longitude if possible. Range or Bearing only messages can also be included.

DLE:	byte;	0x05
STX:	byte;	0x02
MessageLength:	byte;	Depends on number of traffic items
MessageLengthXOR:	byte;	Depends on number of traffic items
MessageType:	byte;	0x06
MessageRate:	byte;	0x01
MessageCount:	byte;	0 or 1, alternating with every transmission
MessageVersion:	byte;	0x01
Traffic mode:	byte;	See description below
Number of traffic:	byte;	Number of traffic in this transmission (0 to 32)
Number of messages:	byte;	Number of messages (1,2,3 or 4)
Message number:	byte;	Number of this message (starts with “1” or zero if none).

Followed by: 0 to 7 messages of a traffic item (32 bytes each)

Latitude:	longint;	in degrees / 180000 or Range in meters
Longitude:	longint;	in degrees / 180000 or Bearing in 10^{th} degree
Altitude:	longint;	in feet. 0X80000000 if not known
Track:	smallint;	in 10^{th} degrees. -1 if not known
Speed:	smallint;	in Km/h. -1 if not known
Vertical Speed:	longint;	in feet/minute. Positive is climbing.
Callsign:	string[6]	Callsign. 0X00 in first location if not known
Source:	byte;	Source of traffic information, see below
Threat level:	byte;	See below. 0 if not known
Resolution:	byte;	See below. 0 if no resolution system
Aircraft category:	byte;	See below. 0 if not known
Traffic ID:	byte;	Number of traffic message in total transmission

If no traffic is available, message length is set to 1 and a blank data portion with 9 bytes of value 0x00 is sent.

Traffic mode:
0 – Traffic items are unsorted
1 -- Traffic items are sorted by distance
2 – Traffic items are sorted by threat level

Traffic source:
0 - Unknown
1 - TCAS or TIS (ARINC 429 or other data feed)
2 - PCAS
3 - FLARM

4 - ADS-B
5 - ADS-B 1090 ES
6 - Unspecified source giving at least lat/long of traffic location
7 - Unspecified source giving range only
8 - Unspecified source giving bearing only

Threat level
0 – Unknown
1 – None
2 – Mild
3 – High
4 – Danger

Resolution
0 – Unknown
Bit 0 set – pull up
Bit 1 set – push down
Bit 2 set – bank right
Bit 3 set – bank left
Bit 4 set – speed up
Bit 5 set – slow down
Bit 6 set – Add “sharp” to resolution
Bit 7 set – Threat resolution available, do nothing

Aircraft category

Aircraft categories are based on DO-260B and defined as follows:

0 - No defined category, emitter set “A”
1 - Light
2 - Small
3 - Large
4 - High vortex large
5 - Heavy
6 - High performance
7 - Rotor craft
8 - No defined category, emitter set “B”
9 - Glider
10 - Lighter than air
11 - Parachutist
12 - Ultralight
13 - Reserved
14 - UAV
15 - Space
16 - No defined category, emitter set “C”
17 - Surface, emergency vehicle
18 - Surface, service vehicle
19 - Point obstacle
20 - Line obstacle

21..23 -	Undefiend
24 -	No defined category, emitter set “D”
25..31	Undefined

255 -	This traffic item does not have any category identification

Message 10: Engine data

This message is sent at a rate typically from 1 to 10 times per second depending on implementation. Each engine has one message.

DLE:	byte;	0x05
STX:	byte;	0x02
MessageLength:	byte;	Depends on message content
MessageLengthXOR:	byte;	Depends on message content
MessageType:	byte;	0x0A
MessageRate:	byte;	As required
MessageCount:	byte;	As required
MessageVersion:	byte;	0x01
Engine number:	byte;	1..Number of engines
Engine type:	byte;	0 – Piston 1 – Turbine

For Combustion engines:

Number of EGT:	byte;	
Number of CHT:	byte;	
RPM:	word;	Revolutions / minute
PULSE:	word;	AUX pulse/RPM value
OIL pressure 1:	word;	In 10^{th} of a millibar (Main oil pressure)
OIL pressure 2:	word;	In 10^{th} of a millibar (optional second oil pressure)
Fuel pressure:	word;	In 10^{th} of a millibar
Coolant temperature:	smallint;	In degrees C
OIL temperature 1:	smallint;	In degrees C
OIL temperature 2:	smallint;	In degrees C
AUX temperature 1:	smallint;	In degrees C
AUX temperature 2:	smallint;	In degrees C
AUX temperature 3:	smallint;	In degrees C
AUX temperature 4:	smallint;	In degrees C
Fuel flow:	word;	In 10^{th} liters/hour
AUX flow:	word;	In 10^{th} liters/hour
Manifold pressure:	word;	In 10^{th} of a millibar
Boost pressure:	word;	In 10^{th} of a millibar
Inlet temperature:	smallint;	In degrees C
Ambient pressure:	word;	In 10^{th} of a millibar (intake air pressure)
EGT:	smallint;	In degrees C - Repeated for each EGT
CHT:	smallint;	In degrees C – Repeated for each CHT

For turbine engines:

Inlet temperature:	smallint;	In degrees C
N1	longint;	RPM
N2	longint;	RPM
Exhaust temperature:	smallint;	In degrees C
OIL pressure 1:	word;	In 10^{th} of a millibar (Main oil pressure)

OIL pressure 2:	word;	In 10th of a millibar (optional second oil pressure)
Fuel pressure:	word;	In 10th of a millibar
OIL temperature 1:	smallint;	In degrees C
OIL temperature 2:	smallint;	In degrees C
AUX temperature 1:	smallint;	In degrees C
AUX temperature 2:	smallint;	In degrees C
AUX temperature 3:	smallint;	In degrees C
Fuel flow:	word;	In 10th liters/hour
Ambient pressure:	word;	In 10th of a millibar (intake air pressure)
Padding:	word;	Set to zero. Used to align checksum.

Either engine type is followed by:

Checksum	longint;	CRC32

Message 11: Fuel levels

This message is sent at a rate of 1 times per second.

DLE:	byte;	0x05
STX:	byte;	0x02
MessageLength:	byte;	Depends on message content
MessageLengthXOR:	byte;	Depends on message content
MessageType:	byte;	0x0B
MessageRate:	byte;	1
MessageCount:	byte;	0 – 1 alternating
MessageVersion:	byte;	0x01
Number of tanks:	longint;	Longint used for alignment purposes

For each tank:

Level:	longint;	In 10th liters
Tank type:	byte;	See below
Tank on:	byte;	See below
Tank sensors:	word;	See below

Followed by:

Checksum	longint;	CRC32

Tank type:
0: physical tank with a level sender
1: virtual tank, level is calculated from fuel flow and starting value
2: virtual tank, level is calculated from flight time and starting value
Other values may be used as required by implementer

Tank on:
0: Tank is off
1: Tank is on
2: Unknown

Tank sensors:
This item can be used by implementer as needed. It is recommended to use value 0xFFFF if this item is not used.

Message 30: Navigation

This message is sent typically at a rate of 1 per second.

DLE:	byte;	0x05	
STX:	byte;	0x02	
MessageLength:	byte;	0x2C	;56 bytes – 12 following MessageVersion
MessageLengthXOR:	byte;	0xD3	
MessageType:	byte;	0x04	
MessageRate:	byte;	0x02	
MessageCount:	byte;	Message Count within current second	
MessageVersion:	byte;	0x01	
Flags:	word;	NAV validity flags, see below	
HSISource:	byte;		
VNAVSource:	byte;		
APMode:	byte;	0 = not engaged, 1 = engaged	
Padding:	byte;		
HSINeedleAngle:	smallint;	Relative HSI needle angle +1800 to -1800. 0 = up	
HSIRoseHeading:	word;	0-3599	
HSIDeviation:	smallint;	-4096 to 4095 for full deflection	
VerticalDeviation:	smallint;	-4096 to 4095 for full deflection	
HeadingBug:	smallint;	0-3599	
AltimeterBug:	longint;	in feet	
WPDistance:	longint;		
WPLatitude:	longint;	MGL format, 180000 per degree	
WPLongitude:	longint;	MGL format, 180000 per degree	
WPTrack:	smallint;	0-3599	
VOR1Radial:	smallint;	0-3599	
VOR2Radial:	smallint;	0-3599	
DME1:	word;	in 0.1 Km steps	
DME2:	word;	in 0.1 Km steps	
ILSDeviation:	smallint;	-4096 to 4095 for full deflection	
GSDeviation:	smallint;	-4096 to 4095 for full deflection	
GLSHorizontalDeviation:	smallint;	-4096 to 4095 for full deflection	
GLSVerticalDeviation:	smallint;	-4096 to 4095 for full deflection	
Padding:	word;		
Checksum	longint;	CRC32	

Flag bits are:

0	HSI valid
1	VNAV valid
2	Waypoint valid
3	Autopilot engaged
4	VOR1 valid
5	VOR2 valid
6	DME1 valid
7	DME2 valid
8	ILS valid

9 GS valid
10 GLS valid

All angular values are in 10th of a degree. Deviations range from -4096 to +4095 for full needle deflection.

HSI Nav source
0 Vectors (heading bug)
1 GPS waypoint navigation
2 VOR navigation
3 ILS

VNAV source
0 Altitude bug
1 Glide slope

AP Mode is split into 2 nibbles of 4 bits. The lower 4 bits shows vertical mode, the upper 4 bits shows horizontal mode.

Horizontal mode:
0 Heading bug
1 HSI
Vertical mode:
0 Set Altimeter bug to current altitude, then follow Altimeter bug
1 Altimeter bug
2 Vertical speed hold
3 Flight plan vertical NAV
4 Pitch attitude hold
5 Vertical mode suspended, use only horizontal NAV
6 Air speed hold

Checksum calculation

Checksum calculation is done from the first byte following MessageLengthXOR to the last data byte before the checksum. Here is a sample source in C which uses fast table lookup CRC calculation. The table can be calculated on startup or can be pre-calculated and stored in ROM.

Header File

```
// CRCdemo.h

protected:
    ULONG crc32_table[256];  // Lookup table array
    void Init_CRC32_Table();  // Builds lookup table array
    ULONG Reflect(ULONG ref, char ch);  // Reflects CRC bits in the
lookup table
    int Get_CRC(CString& text);  // Creates a CRC from a text string
```

Source File

```
// CRCdemo.cpp

void CRCdemo::Init_CRC32_Table()
{// Call this function only once to initialize the CRC table.

    // This is the official polynomial used by CRC-32
    // in PKZip, WinZip and Ethernet.
    ULONG ulPolynomial = 0x04c11db7;

    // 256 values representing ASCII character codes.
    for(int i = 0; i <= 0xFF; i++)
    {
        crc32_table[i]=Reflect(i, 8) << 24;
        for (int j = 0; j < 8; j++)
            crc32_table[i] = (crc32_table[i] << 1) ^ (crc32_table[i] & (1 <<
31) ? ulPolynomial : 0);
        crc32_table[i] = Reflect(crc32_table[i], 32);
    }
}

ULONG CRCdemo::Reflect(ULONG ref, char ch)
{// Used only by Init_CRC32_Table().

    ULONG value(0);

    // Swap bit 0 for bit 7
```

```
    // bit 1 for bit 6, etc.
    for(int i = 1; i < (ch + 1); i++)
    {
        if(ref & 1)
            value |= 1 << (ch - i);
        ref >>= 1;
    }
    return value;
}

int CRCdemo::Get_CRC(CString& text)
{ // Pass a text string to this function and it will return the CRC.

    // Once the lookup table has been filled in by the two functions above,
    // this function creates all CRCs using only the lookup table.
    // Note that CString is an MFC class.
    // If you don't have MFC, use the function below instead.

    // Be sure to use unsigned variables,
    // because negative values introduce high bits
    // where zero bits are required.

    // Start out with all bits set high.
    ULONG  ulCRC(0xffffffff);
    int len;
    unsigned char* buffer;

    // Get the length.
    len = text.GetLength();
    // Save the text in the buffer.
    buffer = (unsigned char*)(LPCTSTR)text;
    // Perform the algorithm on each character
    // in the string, using the lookup table values.
    while(len--)
        ulCRC = (ulCRC >> 8) ^ crc32_table[(ulCRC & 0xFF) ^ *buffer++];
    // Exclusive OR the result with the beginning value.
    return ulCRC ^ 0xffffffff;
}
```

Revision history

1 - Internal release

2 - Internal release

3 - Changed spec for length byte in header to accommodate 256 byte data pages with private 8 byte header. Added Traffic file (message 06).

4 - Added message 10 (Engine data), Message 11 (Fuel level)

5 - Internal release

6 - Redefined message 04 as “various inputs”, Changed message 06 to message 05. Dropped autopilot message (data now included in Navigation message). Defined message 04 and message 30. Clarified length byte.