Serial communication Documentation

Date:	September, 2023
Version:	20230919
Status:	Updated
Owner:	AMT Netherlands

Engine Control Unit documentation
Serial communication

Revision	Date	Major modifications	Chapter
20110615	Jun, 2011	Init document. Split from master document	All
20111207	Dec, 2011	Added serial communication input values for OPS & THR	2.2.2, 2.2.3
20120118	Jan, 2012	Changed offset throttle from 16 to 32	
20120306	Mar, 2012	Added sequence of sending serial data stream	
20121222	Dec,2012	Added new engine to software	
20140617	Jun,2014	Change DATA1 information	
20150203	Feb, 2015	Bits 5&6 changed for max RPM	2.2.2
20150620	Jun,2015	Added new engine to software	All
20160412	Apr, 2016	Corrected schematic ECU-PC-EDT and table bit status	2.2.2
20170411	Apr, 2017	Correction of document 20150203	2.2.2
20181210	Dec, 2018	Scope of document change to only V2 and V3 ECU types. Change calculation values different engine types.	1.3 2.2
20190527	May, 2019	Minor text Additional protocol information	2.2.1 2.3.1
20190604	Jun, 2019	Adding text sequence normal, alternating and error data sets	2.1.1
20201124	Nov, 2020	Time frame complete package of data changed	2.3.1
20201229	Dec, 2020	C-source for decomposing data stream	
20210706	Jul, 2021	Extra explanation of data byte 1 during error mode	2.2.2
20230404	Apr, 2023	New drawing and picture of USB cable & EDT connection	
20230919	Sept, 2023	Update 2.2.2, 2.2.3 and 2.2.4 for more clarity	2.2.2, 2.2.3, 2.2.4

Engine Control Unit documentation
Serial communication

Table of contents

Engine Control Unit documentation
Serial communication

1 Introduction

1.1 Objective of this document

This document contains the information of the serial communication implemented in the ECU hardware.
The target group of this document is AMT-Netherlands and third parties.

1.2 Cohesion of documents

Together with the Engine Control Unit (ECU) specifications and the communication specification a complete set of specification documents is formed. Development is and will be based upon this set of documents.

1.3 Scope of this document

This document is suited for ECU types V2(firmware Vx35…Vx39, Vx80) and V3 (Vx01…Vx25 or Vx41…Vx99) of AMT Netherlands. Wherever possible any deviations for other than AMT-Netherlands developments will be indicated.

1.4 About this document

This document starts with general requirements for the system and is followed by sections that deal with the signals and features of the ECU.

1.5 Links to other documents

The following sources serve as input for this document:

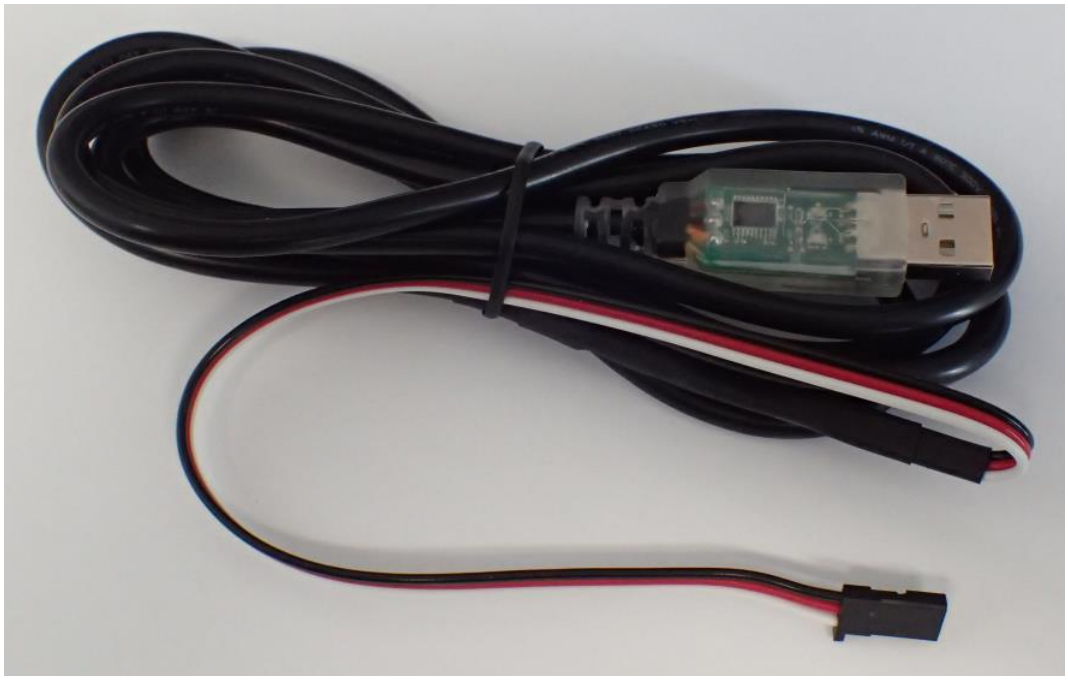
- AMT requirements..... AR
- Specification: Hardware ECU..... SH
- Specification: Software ECU ..... SS

2 Serial control

2.1 General

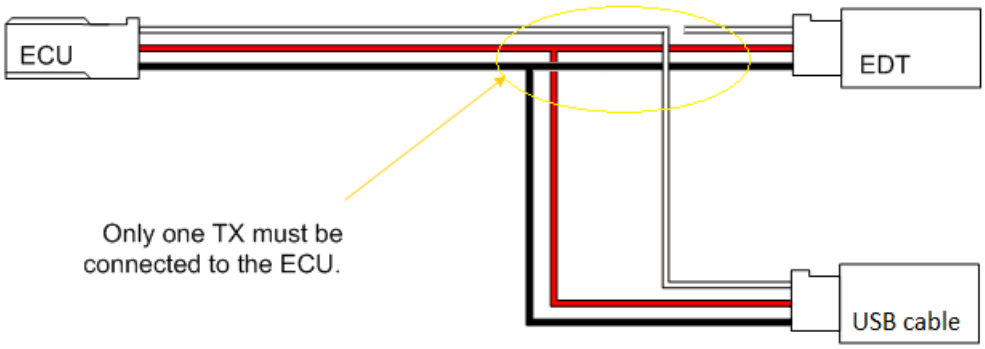
Besides receiver PWM pulses and analog control a third option is possible, via a serial protocol. The EDT connection is used to send data to the ECU with OPS and THR information.
For safety reasons a constant stream of data is required. If this stream is missing for a period, adjustable by a parameter, the turbine will be stopped.

With the windows TMC application and special USB cable the ECU/EPV can be programmed and controlled.



USB cable for serial communication with the TMC application.

Below a wire diagram is given which must be used when in parallel an Engine Data Terminal (EDT) and serial communication device is used.



2.2 EDT data

The ECU reports its status and condition via a serial protocol. This serial protocol is based on the industrial standard RS232.
After power up the ECU transmitted the software version, the software date and a number of settings. This is followed by a the normal serial data stream.

2.2.1 General specifications

Item	Description
Level	Standard RS232 level -12V to 12V
Baud rate	Standard the ECU setting is 2400. This gives an average of 48 bytes per second. Other settings: 2400-4800-9600-19200-38400-57600-115200. If firmware Vx01-Vx25 or Vx43-Vx99 is used the lowest baud rate is 9600.
Protocol setup	8 data bits, no parity, 1 stop-bit
Data stream*	0xFF,{data1},{data2},{data3},{data4},{data5}

*Value of the data bytes 1 to 5 is always between 0 and 0xFE (254). After 10 data sets of normal information an alternating data set is send. If the ECU is in error mode only the error data set is send and the alternating data set is skipped.

2.2.2 Normal information data set

<table>
<tr><th>Byte</th><th>Unit</th><th>Description</th></tr>
<tr><td>Leader</td><td></td><td>Value always 0xFF (255)</td></tr>
<tr><td>Data 1 (Status)</td><td></td><td>This data byte describes the state of the ECU and which type of engine is installed.

<table>
<tr><th colspan="8">Bits</th><th>Description</th></tr>
<tr><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th><th></th></tr>
<tr><td colspan="9">Switch position</td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>0</td><td>0</td><td>1</td><td>Operator Switch in Emergency stop mode</td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>0</td><td>1</td><td>0</td><td>Operator Switch in Auto stop mode</td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>1</td><td>0</td><td>0</td><td>Operator Switch in Running mode</td></tr>
<tr><td colspan="9">Engine status</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>S</td><td>S</td><td>S</td><td>No start clearance / Error[3]</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>S</td><td>S</td><td>S</td><td>Start clearance</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>S</td><td>S</td><td>S</td><td>Starting</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>S</td><td>S</td><td>S</td><td>Started up</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>S</td><td>S</td><td>S</td><td>Idle calibration</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>S</td><td>S</td><td>S</td><td>Fully operation running turbine</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>S</td><td>S</td><td>S</td><td>Maximum RPM reached</td></tr>
<tr><td colspan="9">X = Status of engine
S = state as in Operator Switch
001 = emer. Stop
010 = auto stop
100 = run</td></tr>
</table>
</td></tr>
<tr><td>Data 2 (RPM value)</td><td>RPM</td><td>RPM = value * 500 (Engine ID: PEGASUS, OLYMPUS, TITAN, NIKE, LYNX)
RPM = value * 700 (Engine ID: MERCURY)</td></tr>
<tr><td>Data 3 (EGT value)</td><td>˚C</td><td>EGT = value * 4.6 – 50</td></tr>
<tr><td>Data 4 (Throttle setting)</td><td>%</td><td>THR = value / 2</td></tr>
<tr><td rowspan="2">Data 5 (Vout value)</td><td rowspan="2">Volt</td><td>VOUT[1] = value * 6.25 / 255</td></tr>
<tr><td>VOUT[2] = value * 8.30 / 255</td></tr>
</table>

[1] Engine ID: MERCURY , PEGASUS

[2] Engine ID: OLYMPUS, TITAN, NIKE, LYNX, ORION

[3] If a fully operational running turbine stops with an error, the engine status will be 00000100 meaning ‘No start clearance’. After S is switched back from 100 to 001 and an error is active, the ECU will respond by setting this byte to 00000000 and start sending ‘Error information data sets’ (See 2.2.3).

2.2.3 Error information data set

<table>
<tr><th>Byte</th><th>Unit</th><th>Description</th></tr>
<tr><td>Leader</td><td></td><td>Value always 0xFF (255)</td></tr>
<tr><td>Data 1 (Status)</td><td></td><td>In the ‘Error information data set’ this byte is always zero.</td></tr>
<tr><td></td><td></td><td>
<table>
<tr><th colspan="8">Bits</th><th>Description</th></tr>
<tr><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th><th></th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>Error</td></tr>
</table>
</td></tr>
<tr><td>Data 2 (Error code)</td><td></td><td>
RPM value is overruled with the error code of the ECU. Next table gives the relation between error code and the bits which are set. Multiple error can occur. In this case more then one bit is set.
<table>
<tr><th colspan="8">Bits</th><th>Description</th></tr>
<tr><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th><th></th></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>1</td><td>rpm low</td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>1</td><td>X</td><td>switch channel not present</td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>1</td><td>X</td><td>X</td><td>throttle channel not present</td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>1</td><td>X</td><td>X</td><td>X</td><td>EGT error</td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>1</td><td>X</td><td>X</td><td>X</td><td>X</td><td>rpm high</td></tr>
<tr><td>X</td><td>X</td><td>1</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>supply low</td></tr>
<tr><td>X</td><td>1</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>supply low for Auto Start System</td></tr>
<tr><td>1</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td></td></tr>
</table>
X = 1 or 0
</td></tr>
<tr><td>Data 3 (EGT value)</td><td>˚C</td><td>EGT = value * 4.6 – 50</td></tr>
<tr><td>Data 4 (Throttle setting)</td><td>%</td><td>THR = value / 2</td></tr>
<tr><td>Data 5 (Vout value)</td><td>Volt</td><td>VOUT[1] = value * 6.25 / 255
VOUT[2] = value * 8.30 / 255</td></tr>
</table>

[1] Engine ID: MERCURY , PEGASUS
[2] Engine ID: OLYMPUS, TITAN, NIKE, LYNX, ORION

2.2.4 Alternate information data set

<table>
<tr><th>Byte</th><th>Unit</th><th>Description</th></tr>
<tr><td>Leader</td><td></td><td>Value always 0xFF (255)</td></tr>
<tr><td>Data 1 (Engine ID)</td><td></td><td>If B2, B1 and B0 are zero (low) then the bits B7 to B3 will indicate which engine is installed.

<table>
<tr><th colspan="8">Bits</th><th>Description</th></tr>
<tr><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th><th></th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>PEGASUS engine ID (older versions)</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>OLYMPUS engine ID (older versions)</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>MERCURY engine ID (older versions)</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>MERCURY engine ID</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>PEGASUS engine ID</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>OLYMPUS engine ID</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>TITAN engine ID</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>NIKE engine ID</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>LYNX engine ID</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>ORION engine ID</td></tr>
</table></td></tr>
<tr><td rowspan="2">Data 2 (idle voltage)</td><td rowspan="2">Volt</td><td>PWOMIN[1] = value * 6.25 /255</td></tr>
<tr><td>PWOMIN[2] = value * 8.30 / 255</td></tr>
<tr><td rowspan="2">Data 3 (max rpm voltage)</td><td rowspan="2">Volt</td><td>PWOMAX[1] = value * 6.25 / 255</td></tr>
<tr><td>PWOMAX[2] = value * 8.30 / 255</td></tr>
<tr><td>Data 4 (ECU info)</td><td></td><td>Bitwise information** will be alternated with the value 254 for compatibility reasons:

Bit 0: 0 = None BLDC, 1 = BLDC version of the ECU

Other bits are not defined yet.</td></tr>
<tr><td rowspan="4">Data 5 (battery voltage)</td><td rowspan="4">Volt</td><td>VSUP[1]= 7 + (value * 6.25 / 255)</td></tr>
<tr><td>VSUP[2] = 7 + (value * 9.30 / 255)</td></tr>
<tr><td>VSUP[3] = 18+ (value * 12.00 / 255)</td></tr>
<tr><td>VSUP[4] = 5 + (value / 10)</td></tr>
</table>

[1] Engine ID: MERCURY, PEGASUS (firmware Vx35…Vx39)
[2] Engine ID: OLYMPUS, TITAN, NIKE, ORION (firmware Vx35…Vx39 and Vx90…Vx99)
[3] Engine ID: LYNX (firmware Vx40…Vx42)
[4] Engine ID: ALL (firmware Vx01…Vx25 and Vx43…Vx89)

2.2.5 ECU set-up data set

Byte	Unit	Description
Leader		Value always 0xFF (255)
Data 1 (ECU set-up)		Value always 0x05 (5)
Data 2 (high byte)*	ms	Input pulse width information from the switch channel Pulse width minimum: PWmin = ((256 * PWTH)+PWTL)-62464
Data 3 (low byte)*		
Data 4 (pulse difference)*	ms	PWmax = PWmin + PWIDIFF * 16
Data 5…17		A string of 12 bytes is transmitted (ASCII) which containing the software version and the production date. Format: "X.NN YYMMDD " (old) “XX.NN YYMMDD “ (new) Software version and engine type. 1.NN: Pegasus engine (obsolete) 2.NN: Pegasus engine (obsolete) 3.NN: Olympus engine 4.NN: Pegasus engine 5.NN: Mercury engine 6.NN: Olympus engine 7.NN: Pegasus engine 8.NN: Mercury engine 9.NN: Olympus engine 10.NN: Titan engine 12.NN: Nike engine 14.NN: Lynx engine 16.NN: Orion engine

*Only to be used when Radio Controlled pulse method is switch on as input value

2.3 Serial communication implementation

2.3.1 Protocol implementation (firmware: Vx25…Vx42)

0xFF	OPS	THR	OPS	THR		0xFF	OPS	THR	OPS	THR	

⇧

Pause between messages.

The message will start with a 255 byte value to indicate that new OPS (switch) and THR (throttle) info is coming. OPS and THR will be sending twice and the ECU software will determine if both bytes of the OPS and THR are the same. When these bytes are not the same the message is ignored and the error counter increased.

To make sure data will be excepted without errors next implementation must be guaranteed. This means that a serial control data stream must be sent directly after the last byte of the EDT data stream is received.

EDT data stream

0xFF	Data1	Data2	Data3	Data4	Data5		0xFF	Data1	Data2	Data3

Serial control data stream

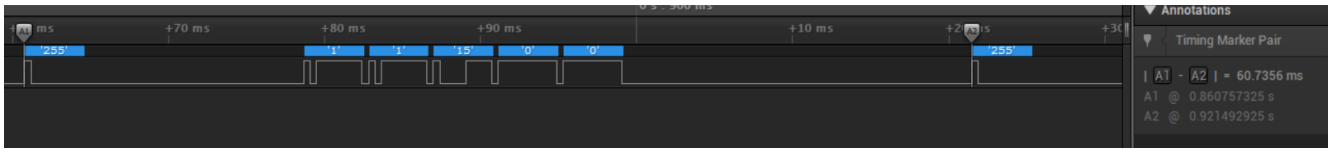
	0xFF	OPS	THR	OPS	THR	

*Max. 65msec

Calculation:

- Databyte has 1*startbit, 8*databits, no parity and 1*stopbit so in total 10 bits.
- Time per byte: (1 / 2400) * 10 = 4,16msec.
- Minimum time serial data stream: 5(data bytes) x 10bits = 50 bits or 5(data bytes) * 4,16 = 20,8msec.

Measuring a frame of 5 bytes at 2400Baud takes 60msec to complete. Delays between the bytes causes by the sequential software implementation.



Screenshot real-life frame (Vx38).

*Frame time is valid for software versions Vx36 and Vx38. Vx37 frame time is 63ms. For older firmware versions the frame time can differ and must be measured if this frame time must be known.

2.3.2 Protocol implementation (firmware: Vx01-Vx25 or Vx43-Vx99)

Due to a different implementation the pause between the messages is almost gone. Also, receiving information via the serial communication channel will be handled asynchrony to the send information.
Make sure that the control message (0xFF, OPS, THR, OPS, THR) is send at least every 100ms.

2.3.3 General specifications

Item	Description
Level	Standard RS232 level -12V to 12V
Baud rate	Standard the ECU setting is 2400. This gives an average of 48 bytes per second. Other settings are 2400-4800-9600-19200-38400-57600-115200.
Protocol setup	8 data bits, no parity, 1 stop-bit
Data stream*	0xFF,{data1},{data2},{data3},{data4}

*Value of the data bytes 1 to 4 is always between 0 and 0xFE (254).

2.3.4 Engine control data set

<table>
<tr><th>Byte</th><th>Unit</th><th>Description</th></tr>
<tr><td>Leader</td><td></td><td>Value always 0xFF (255)</td></tr>
<tr><td>Data 1 (OPS Status)</td><td></td><td>This data byte describes the state of the Operation Switch (OPS).
<table>
<tr><th colspan="8">Bits</th><th>Description</th></tr>
<tr><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th><th></th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>Operator Switch in Emergency stop mode</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>Operator Switch in Auto-stop mode</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>Operator Switch in Running mode</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>CTF (special functions)*</td></tr>
</table></td></tr>
<tr><td>Data 2 (THR value)</td><td>%</td><td>This data byte describes the throttle setting.

Data2 = (THR * 2) + 32

Example: 50% → (50 * 2) + 32 = 132</td></tr>
<tr><td>Data 3 (OPS status)</td><td></td><td>See Data 1</td></tr>
<tr><td>Data 4 (THR value)</td><td>%</td><td>See Data 2</td></tr>
</table>

*If OPS value is 145 (CTF) during running mode, the engine will stop after the serial delay time (address 192) is elapsed.

2.3.5 Sample code

Below an example how the source code could be. Function PutC is a function which sends out a byte via the serial port opened in advanced.

```
Void SendCommandToECU( unsigned char switch_value, unsigned char throttle_value )
{
        // Make sure that this is send at least every 50ms
        PutC( 255 );                                    // Send this value via RS232 to ECU
        PutC( switch_value );
        PutC( throttle_value );
        PutC( switch_value );
        PutC( throttle_value );
}
```

Function call: SendCommandToECU(17, 32); // OPS = Emergency stop, THR = 0%

2.3.6 Sample code decompose data stream

```
void ConvertSerialData(){
  // Data received on UART
  if( comm_recieved_pointer != comm_convert_pointer ){ // Check if new data is available in buffer
    comm_data_byte = comm_data[ comm_convert_pointer ];// Get data out of buffer
    comm_data[ comm_convert_pointer ] = 0;             // Clear mem in buffer
    comm_convert_pointer++;                           // Increase buffer pointer after reading
    if( comm_data_byte == DATA_MARKER ){              // Check if data is marker byte
      EDT_info_pointer = DATA_BYTE_1;                 // Reset info pointer because start of new frame
    }else
    {
      if( EDT_info_pointer == DATA_BYTE_1 ){          // First byte in frame
        EDT_engine_info = false;                      // Engine info is deselected
        if( comm_data_byte == MERCURY ){              // Engine Mercury detected
          EDT_engine_name = MERCURY;                  // Store engine type
          EDT_engine_info = true;                     // Following bytes are alternating data
          EDT_RPM_factor  = RPM_FACTOR_700;           // Set correct RPM factor for this engine
          EDT_VOUT_factor = VOUT_FACTOR_625;          // Set correct Vout factor for this engine
        }
        if( comm_data_byte == PEGASUS ){              // Engine PEGASUS detected
          EDT_engine_name = PEGASUS;
          EDT_engine_info = true;
          EDT_RPM_factor  = RPM_FACTOR_500;
          EDT_VOUT_factor = VOUT_FACTOR_625;
        }
        if( comm_data_byte == OLYMPUS ){              // Engine OLYMPUS detected
          EDT_engine_name = OLYMPUS;
          EDT_engine_info = true;
          EDT_RPM_factor  = RPM_FACTOR_500;
          EDT_VOUT_factor = VOUT_FACTOR_830;
        }
        if( comm_data_byte == TITAN ){                // Engine TITAN detected
          EDT_engine_name = TITAN;
          EDT_engine_info = true;
          EDT_RPM_factor  = RPM_FACTOR_500;
          EDT_VOUT_factor = VOUT_FACTOR_830;
        }
        if( comm_data_byte == NIKE ){                // Engine Nike detected
          EDT_engine_name = NIKE;
          EDT_engine_info = true;
          EDT_RPM_factor  = RPM_FACTOR_500;
          EDT_VOUT_factor = VOUT_FACTOR_830;
        }
        if( comm_data_byte == LYNX ){                // Engine Lynx detected
          EDT_engine_name = LYNX;
          EDT_engine_info = true;
          EDT_RPM_factor  = RPM_FACTOR_500;
          EDT_VOUT_factor = VOUT_FACTOR_830;
          }
        if( comm_data_byte == ORION ){               // Engine Orion detected
          EDT_engine_name = ORION;
          EDT_engine_info = true;
          EDT_RPM_factor  = RPM_FACTOR_500;
          EDT_VOUT_factor = VOUT_FACTOR_830;
          }
        if( comm_data_byte == VERSION ){             // Next byte in frame are version related
          EDT_info_pointer = DATA_VERSION;
        }
        if( comm_data_byte == 0 ){                   // First byte signals that an error is occurred
          if( !EDT_ECU_error ){
            EDT_ECU_error    = true;
            EDT_ECU_err_info = 0;
          }
        }else{
          if( !EDT_engine_info ){
            EDT_ECU_info = comm_data_byte;           // First byte contains ECU status info
            if(( comm_data_byte & 0x07 ) == 1 ){     // Bit 0 is set then OPS switch is in
              EDT_ECU_info_STOP  = true;             // EMER.-STOP
              EDT_ECU_info_AUTO  = false;
              EDT_ECU_info_START = false;
            }
```

```
      if(( comm_data_byte & 0x07 ) == 2 ){       // Bit 1 is set then OPS switch is in AUTO.-STOP
        EDT_ECU_info_STOP  = false;
        EDT_ECU_info_AUTO  = true;
        EDT_ECU_info_START = false;
      }
      if(( comm_data_byte & 0x07 ) == 4 ){       // Bit 2 is set then OPS switch is in RUNNING
        EDT_ECU_info_STOP  = false;
        EDT_ECU_info_AUTO  = false;
        EDT_ECU_info_START = true;
      }
      EDT_ECU_error      = false;
  // No error have occur
      EDT_ECU_err_info   = 0;
    }
  }
}else{
  if( EDT_info_pointer == DATA_BYTE_2 ){         // Second byte in frame
    if( EDT_ECU_error ){                         // Check if previous byte was 0 indicating
      EDT_ECU_err_info = comm_data_byte;         // an error. Byte contains error information
    }else{
      if( EDT_engine_info ){                     // Check if alternating data is set
        EDT_PUMP_idle = comm_data_byte;          // Byte contains pump voltage at idle side.
                                                 // VMIN = data * EDT_VOUT_factor [V]
      }else{                                     // else
        EDT_RPM_info  = comm_data_byte;          // byte contains RPM data.
                                                 // RPM = data * EDT_RPM_factor
      }
    }
  }else{
    if( EDT_info_pointer == DATA_BYTE_3 ){       // Thrid byte in frame
      if( EDT_engine_info ){
        EDT_PUMP_max = comm_data_byte;           // Byte contains pump voltage at max side.
                                                 // VMAX = data * EDT_VOUT_factor [V]
      }else{                                     // else
        EDT_EGT_info  = comm_data_byte;          // byte contains EGT data.
                                                 // EGT = ( data * 4.6 ) - 50 [C]
      }
    }else{
      if( EDT_info_pointer == DATA_BYTE_4 ){     // Four byte in frame
        if( EDT_engine_info ){
          EDT_special_info = false;              // Special info
          if( comm_data_byte != 254 ){           // If data isn't equal to 254 (254 for older
            EDT_special_info = true;             // systems) new V3 system
            EDT_vx43_and_higher_counter = 3;     // Software detection
          }else{
            if( EDT_vx43_and_higher_counter == 0 ){
            }else{
              EDT_vx43_and_higher_counter--;
            }
          }
        }else{                                   // else
          EDT_THR_info  = comm_data_byte;        // byte contains THR data.
                                                 // THR = data / 2 [%]
        }
      }else{
        if( EDT_info_pointer == DATA_BYTE_5 ){
          if( EDT_engine_info ){
            // Byte is not used
            if( !EDT_special_info ){             // Check if special info is sent
              EDT_SUPP_info = comm_data_byte;    // else byte contains SUPPLY data.
                                                 // See function to convert SUPPLY data.
            }
            EDT_ECU_error = false;
          }else{                                 // else
            EDT_PUMP_info = comm_data_byte;      // byte contains PUMP data.
                                                 // PUMP = data * EDT_VOUT_factor [V]
          }
          if( !EDT_extra_info ){
            EDT_info_pointer    = DATA_WAIT;     // Go idle until next marker is detected
          }
        }else{
          if( EDT_info_pointer == DATA_BYTE_6 ){
            if( EDT_extra_info ){
```

```
    EDT_extra_data      = comm_data_byte;
  }
  EDT_info_pointer    = DATA_WAIT;
}else{
  // This part is mainly used for version detection of the EDT
  if( EDT_info_pointer == DATA_BYTE_1_VERSION ){
    // High byte of pulse width
    ECU_min_pulse_temp  = (int16)comm_data_byte;
    ECU_min_pulse_temp *= 256;
  }else{
    if( EDT_info_pointer == DATA_BYTE_2_VERSION ){
      // Low byte of pulse width
      ECU_min_pulse_temp  += comm_data_byte;
      ECU_min_pulse_temp  -= 62464;
      ECU_min_pulse_temp  /= 10;
      ECU_min_pulse_width  = ECU_min_pulse_temp;
    }else{
      if( EDT_info_pointer == DATA_BYTE_3_VERSION ){
        // Diff. byte of pulse width
        ECU_min_pulse_temp  = (int16)comm_data_byte;
        ECU_min_pulse_temp *= 16;
        ECU_min_pulse_temp /= 10;
        ECU_min_pulse_diff  = ECU_min_pulse_temp;
      }else{
        if( EDT_info_pointer == DATA_BYTE_4_VERSION ){
          ECU_version_2_digits = FALSE;
          // Major number of version ECU N.nn
          comm_data_byte    = comm_data_byte - 48;
          ECU_version_info  = (Int16)comm_data_byte;
          ECU_version_info *= 100;

          if( comm_data_byte == 5 || comm_data_byte == 8 ){
            EDT_engine_name  = MERCURY;
            EDT_RPM_factor   = RPM_FACTOR_700;
            EDT_VOUT_factor  = VOUT_FACTOR_625;
            EDT_info_pointer = DATA_BYTE_5_VERSION;
          }else{
            if( comm_data_byte == 4 || comm_data_byte == 7 ){
              EDT_engine_name  = PEGASUS;
              EDT_RPM_factor   = RPM_FACTOR_500;
              EDT_VOUT_factor  = VOUT_FACTOR_625;
              EDT_info_pointer = DATA_BYTE_5_VERSION;
            }else{
              if( comm_data_byte == 3 || comm_data_byte == 6 ){
                EDT_engine_name  = OLYMPUS;
                EDT_RPM_factor   = RPM_FACTOR_500;
                EDT_VOUT_factor  = VOUT_FACTOR_830;
                EDT_info_pointer = DATA_BYTE_5_VERSION;
              }else{
                if( comm_data_byte == 1 || comm_data_byte == 2 ){
                  // Number of version ECU Nn.nn
                  ECU_version_info    *= 10;
                  ECU_version_2_digits = TRUE;
                }else{
                  if( comm_data_byte == 9 ){
                    EDT_engine_name  = TITAN;
                    EDT_RPM_factor   = RPM_FACTOR_500;
                    EDT_VOUT_factor  = VOUT_FACTOR_830;
                    EDT_info_pointer = DATA_BYTE_5_VERSION;
                  }
                }
              }
            }
          }
        }else{
          if( EDT_info_pointer == DATA_BYTE_5_VERSION ){
            if( ECU_version_2_digits ){
              comm_data_byte    = comm_data_byte - 48;
              temp_int16        = (Int16)comm_data_byte;
              temp_int16       *= 100;
              ECU_version_info += (Int16)temp_int16;
              // Number of version ECU nN.nn
              if( comm_data_byte == 0 ){
```

```
                                EDT_engine_name  = TITAN;
                                EDT_RPM_factor   = RPM_FACTOR_500;
                                EDT_VOUT_factor  = VOUT_FACTOR_830;
                              }else{
                                if( comm_data_byte == 2 ){
                                  EDT_engine_name  = NIKE;
                                  EDT_RPM_factor   = RPM_FACTOR_500;
                                  EDT_VOUT_factor  = VOUT_FACTOR_830;
                                }else{
                                  if( comm_data_byte == 4 ){
                                    EDT_engine_name  = LYNX;
                                    EDT_RPM_factor   = RPM_FACTOR_500;
                                    EDT_VOUT_factor  = VOUT_FACTOR_830;
                                  }else{
                                    if( comm_data_byte == 6 ){
                                      EDT_engine_name  = ORION;
                                      EDT_RPM_factor   = RPM_FACTOR_500;
                                      EDT_VOUT_factor  = VOUT_FACTOR_830;
                                    }
                                  }
                                }
                              }
                            }else{
                              if( EDT_info_pointer == DATA_BYTE_6_VERSION ){
                                // Char "." is now sent
                              }else{
                                if( EDT_info_pointer == DATA_BYTE_7_VERSION ){
                                  // Minor number of version ECU n.Nn
                                  temp_int8         = comm_data_byte - 48;
                                  temp_int8        *= 10;
                                  ECU_version_info += (Int16)temp_int8;
                                }else{
                                  if( EDT_info_pointer == DATA_BYTE_8_VERSION ){
                                    // Minor number of version ECU n.nN
                                    temp_int8         = comm_data_byte - 48;
                                    ECU_version_info += (Int16)temp_int8;
                                  }else{
                                    if( EDT_info_pointer == DATA_BYTE_9_VERSION ){
                                      ECU_version_detected = TRUE;
                                      EDT_info_pointer = DATA_WAIT;
                                    }
                                  }
                                }
                              }
                            }
                          }
                        }
                      }
                    }
                  }
                }
              }
            }
          }
        }
      }
    }
    if( EDT_info_pointer != DATA_WAIT ){
       EDT_info_pointer++;                          // Increment info pointer
    }
  }
}
```

2.3.7 Sample code supply voltage

```
Void SupplyData( void ){
  if( EDT_supply_old ){
    data  = (Int32)EDT_SUPP_info;
    data *= VOUT_FACTOR_SUPP;
    data /= 255;
  }else{
    if( EDT_vx43_and_higher_counter != 0 ){
      data = (Int32)EDT_SUPP_info + 50;
    }else{
```

```
      data = (Int32)ECU_SUPP_conversion[EDT_SUPP_info];
    }
  }
}
```

2.3.8 Conversion array supply voltage

```
Int8 const ECU_SUPP_conversion[256] = {
    0, // 0
   73, // 1
   73, // 2
   74, // 3
   74, // 4
   74, // 5
   75, // 6
   75, // 7
   75, // 8
   76, // 9
   76, // 10
   76, // 11
   77, // 12
   77, // 13
   77, // 14
   78, // 15
   78, // 16
   79, // 17
   79, // 18
   79, // 19
   80, // 20
   80, // 21
   80, // 22
   81, // 23
   81, // 24
   81, // 25
   82, // 26
   82, // 27
   82, // 28
   83, // 29
   83, // 30
   83, // 31
   84, // 32
   84, // 33
   84, // 34
   85, // 35
   85, // 36
   85, // 37
   86, // 38
   86, // 39
   86, // 40
   87, // 41
   87, // 42
   87, // 43
   88, // 44
   88, // 45
   89, // 46
   89, // 47
   89, // 48
   90, // 49
   90, // 50
   90, // 51
   91, // 52
   91, // 53
   91, // 54
   92, // 55
   92, // 56
   93, // 57
   93, // 58
   93, // 59
   94, // 60
   94, // 61
   94, // 62
   95, // 63
   95, // 64
```

```
  95, // 65
  96, // 66
  96, // 67
  96, // 68
  97, // 69
  97, // 70
  97, // 71
  98, // 72
  98, // 73
  98, // 74
  99, // 75
  99, // 76
 100, // 77
 100, // 78
 100, // 79
 101, // 80
 101, // 81
 102, // 82
 102, // 83
 102, // 84
 103, // 85
 103, // 86
 103, // 87
 104, // 88
 104, // 89
 104, // 90
 105, // 91
 105, // 92
 105, // 93
 106, // 94
 106, // 95
 106, // 96
 107, // 97
 107, // 98
 108, // 99
 108, // 100
 108, // 101
 109, // 102
 109, // 103
 109, // 104
 110, // 105
 110, // 106
 110, // 107
 111, // 108
 111, // 109
 111, // 110
 112, // 111
 112, // 112
 112, // 113
 113, // 114
 113, // 115
 113, // 116
 114, // 117
 114, // 118
 114, // 119
 115, // 120
 115, // 121
 115, // 122
 116, // 123
 116, // 124
 116, // 125
 117, // 126
 117, // 127
 118, // 128
 118, // 129
 118, // 130
 119, // 131
 119, // 132
 119, // 133
 120, // 134
 120, // 135
 120, // 136
 121, // 137
```

```
121, // 138
122, // 139
122, // 140
122, // 141
123, // 142
123, // 143
123, // 144
124, // 145
124, // 146
124, // 147
125, // 148
125, // 149
125, // 150
126, // 151
126, // 152
126, // 153
127, // 154
127, // 155
128, // 156
128, // 157
128, // 158
129, // 159
129, // 160
129, // 161
130, // 162
130, // 163
130, // 164
131, // 165
131, // 166
132, // 167
132, // 168
132, // 169
133, // 170
133, // 171
133, // 172
134, // 173
134, // 174
135, // 175
135, // 176
135, // 177
136, // 178
136, // 179
136, // 180
137, // 181
137, // 182
137, // 183
138, // 184
138, // 185
138, // 186
139, // 187
139, // 188
140, // 189
140, // 190
140, // 191
141, // 192
141, // 193
141, // 194
142, // 195
142, // 196
142, // 197
143, // 198
143, // 199
143, // 200
144, // 201
144, // 202
144, // 203
145, // 204
145, // 205
145, // 206
146, // 207
146, // 208
147, // 209
147, // 210
```

```
  147, // 211
  148, // 212
  148, // 213
  148, // 214
  149, // 215
  149, // 216
  149, // 217
  150, // 218
  150, // 219
  150, // 220
  151, // 221
  151, // 222
  151, // 223
  152, // 224
  152, // 225
  153, // 226
  153, // 227
  153, // 228
  154, // 229
  154, // 230
  154, // 231
  155, // 232
  155, // 233
  155, // 234
  156, // 235
  156, // 236
  156, // 237
  157, // 238
  157, // 239
  157, // 240
  158, // 241
  158, // 242
  159, // 243
  159, // 244
  159, // 245
  160, // 246
  160, // 247
  160, // 248
  161, // 249
  161, // 250
  162, // 251
  162, // 252
  163, // 253
  163, // 254
  163  // 255
};
```

2.3.9 Defines

```
#define DATA_MARKER             255
#define DATA_BYTE_1             1
#define DATA_BYTE_2             2
#define DATA_BYTE_3             3
#define DATA_BYTE_4             4
#define DATA_BYTE_5             5
#define DATA_BYTE_6             6
#define DATA_BYTE_7             7
#define DATA_BYTE_8             8
#define DATA_VERSION            9
#define DATA_BYTE_1_VERSION     10                  // High byte pulse width
#define DATA_BYTE_2_VERSION     11                  // Low byte pulse width
#define DATA_BYTE_3_VERSION     12                  // Byte of differance pulse
#define DATA_BYTE_4_VERSION     13                  // Major of version number N.nn
#define DATA_BYTE_5_VERSION     14                  // Char .
#define DATA_BYTE_6_VERSION     15                  // Minor of version number n.Nn
#define DATA_BYTE_7_VERSION     16                  // Minor of version number n.nN
#define DATA_BYTE_8_VERSION     17
#define DATA_BYTE_9_VERSION     18
#define DATA_BYTE_10_VERSION    19
#define DATA_BYTE_11_VERSION    20
#define DATA_BYTE_12_VERSION    21
```

```
#define DATA_BYTE_13_VERSION     22
#define DATA_BYTE_14_VERSION     23
#define DATA_WAIT                99

#define EDT_OPS_STOP_POS         0
#define EDT_OPS_AUTOSTOP_POS     1
#define EDT_OPS_START_POS        2
#define EDT_START_CLEARENCE      3
#define EDT_STARTING             4
#define EDT_STARTED_UP           5
#define EDT_CALIBRATION          6
#define EDT_RPM_MAX_SET          7

#define ECU_ERROR_RPM_LOW        0
#define ECU_ERROR_OPS_FAILED     1
#define ECU_ERROR_THR_FAILED     2
#define ECU_ERROR_EGT_HIGH       3
#define ECU_ERROR_RPM_HIGH       4
#define ECU_ERROR_SUPPLY_LOW     5
#define ECU_ERROR_SUPPLY_ASS     6
#define ECU_ERROR_BLANK          7

#define NO_ENGINE                9999
#define VERSION                  5
#define MERCURY                  7
#define PEGASUS                  3
#define OLYMPUS_HP               5
#define OLYMPUS                  6
#define TITAN                    32
#define NIKE                     40
#define LYNX                     48
#define ORION                    56

#define RPM_FACTOR_500           500
#define RPM_FACTOR_700           700

#define VOUT_FACTOR_625          625
#define VOUT_FACTOR_830          830
#define VOUT_FACTOR_SUPP         1000
```


Appendix A Glossary

AMT	Advanced Micro Turbine the Netherlands
AR	AMT requirements
BLDC	Brush Less Direct Current
ECU	Engine Control Unit
EDT	Engine Data Terminal
EGT	Exhaust Gas Temperature
EMC	Electro Magnetic Compatibility
ESD	Electro Static Discharge.
FET	Field Effect Transistor
HAL	Hardware abstraction layer
LCD	Liquid Crystal Display
LED	Light Emitting Diode
MC	Machine Controller
MTBF	Mean Time Between Failures
MTTF	Mean Time To Failure
OH	Old ECU hardware specifications
OPS	Operational switch
OS	Old ECU software specifications
PCB	Printed Circuit Board
PLL	Phase Locked Loop.
RPM	Revolutions per minute
Rx	Receive
Tbd	Too be defined
THR	Throttle
TMC	Turbine Management Control windows application
TSOP	Thin Small Outline Package
Tx	Transmit
UART	Universal Asynchronous Receiver Transmitter
UI	User Interface

