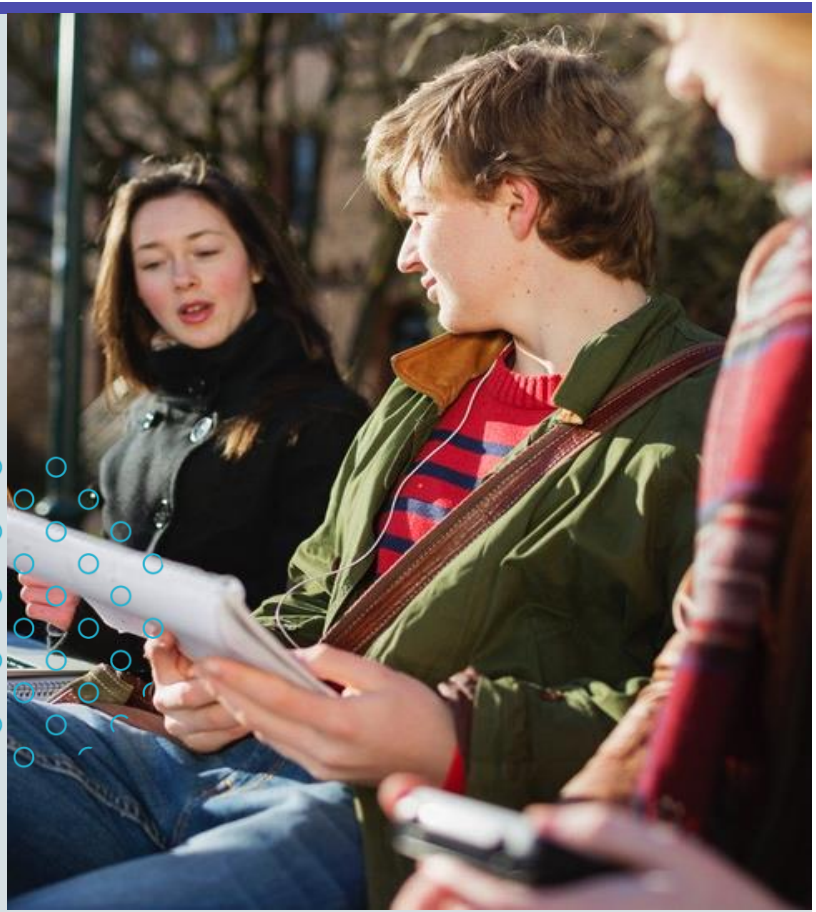
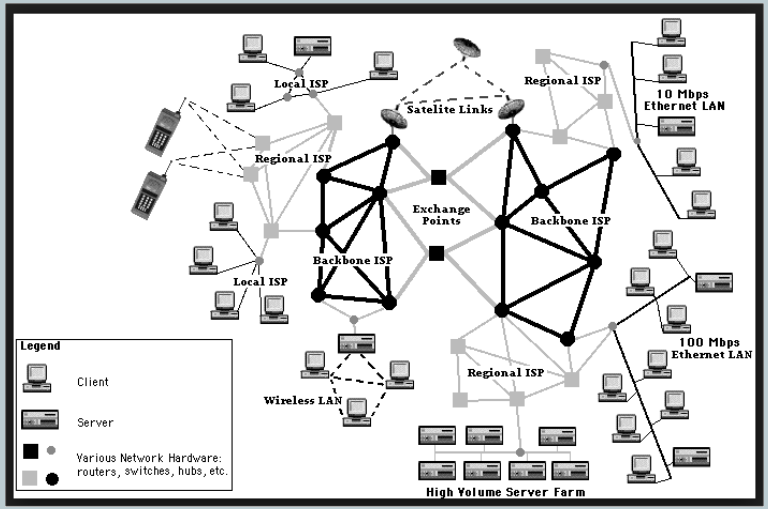
# Data Communication and C# programming

Nils-Olav Skeie
Professor, PhD





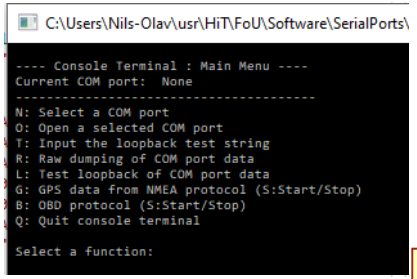

Internet: Network of networks

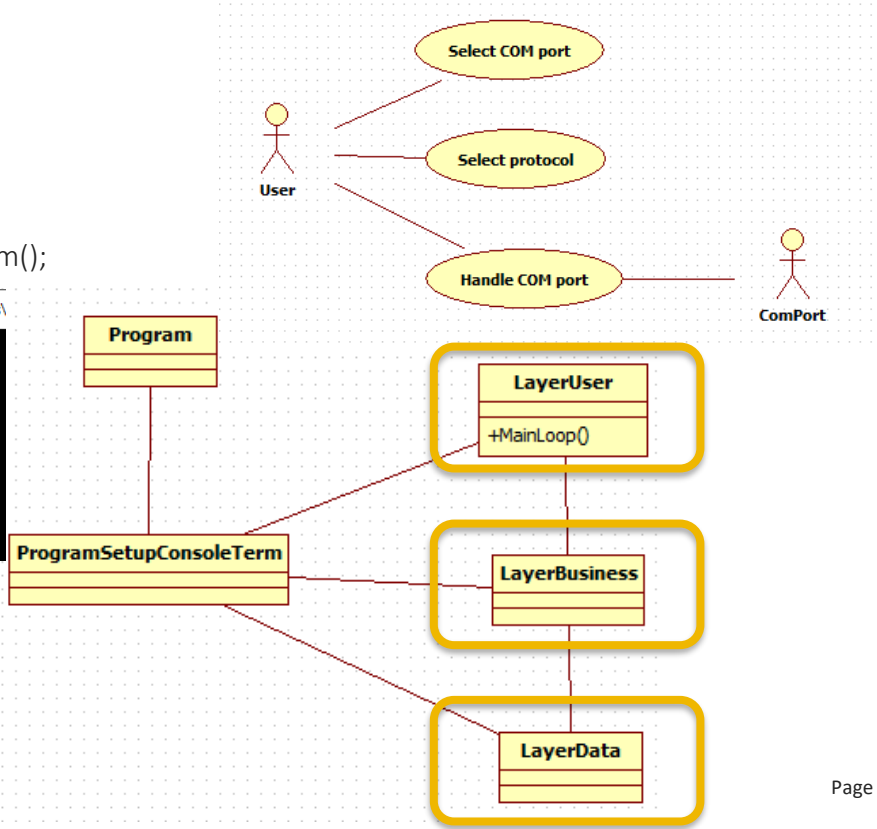
# Agenda

- Console application – 3 tier architecture
- Data Communication,
- Serial port communication,
- Serial port communication in C#,
- Protocols,
- OBD system,
- Network communication,
- Network communication in C#.

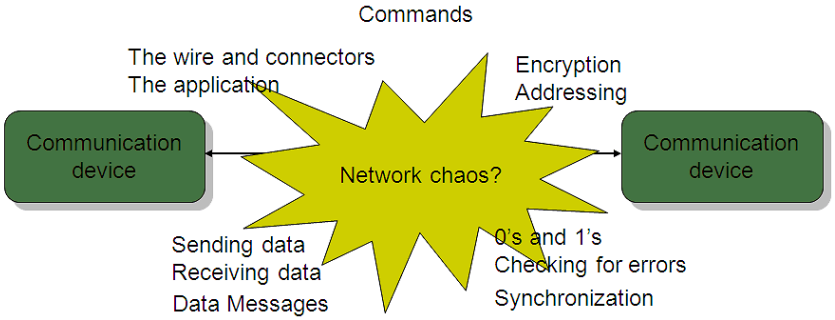
# C# - Terminal Console application – three tier architecture

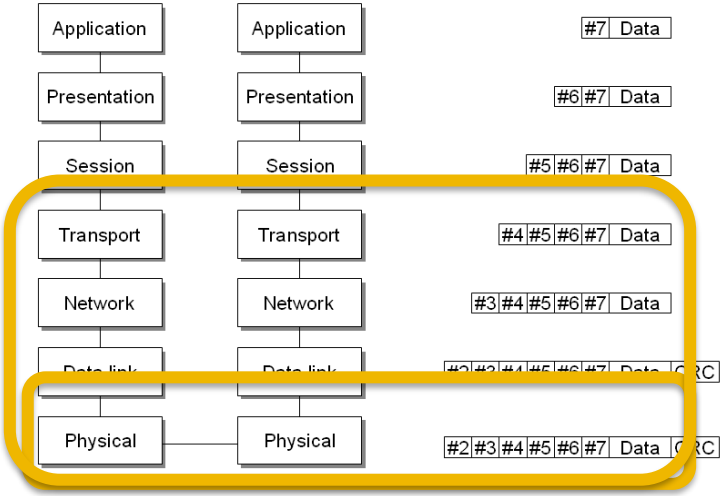
```
class Program
{
    static void Main(string[] args)
    {
        ProgramSetupConsoleTerm  rProgramSetupConsoleTerm;
        rProgramSetupConsoleTerm  = new ProgramSetupConsoleTerm();
    }
}
class ProgramSetupConsoleTerm
{
    LayerData rLayerData;
    LayerBusiness rLayerBusiness;
    LayerUser rLayerUser;

    public ProgramSetupConsoleTerm()
    {
        rLayerData = new LayerData();
        rLayerBusiness = new LayerBusiness(rLayerData);
        rLayerUser = new LayerUser(rLayerBusiness);
        rLayerUser.MainLoop();
    }
}
```



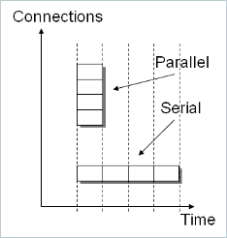
# Data Communication

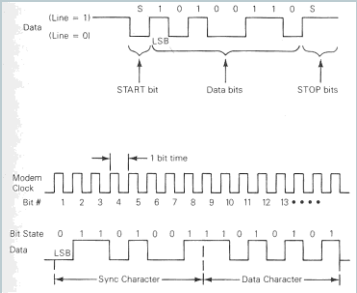


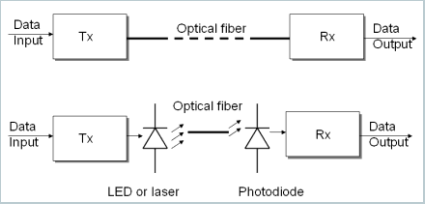

- Communication between two or more computers,
- M2M – Machine to Machine communication,
- Need a protocol – set of rules on how to communicate.

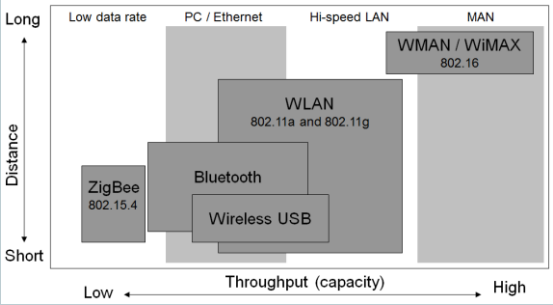
- OSI model (Open System Interconnection).




Add synchronization

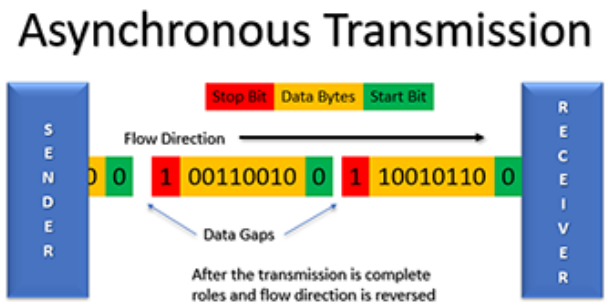


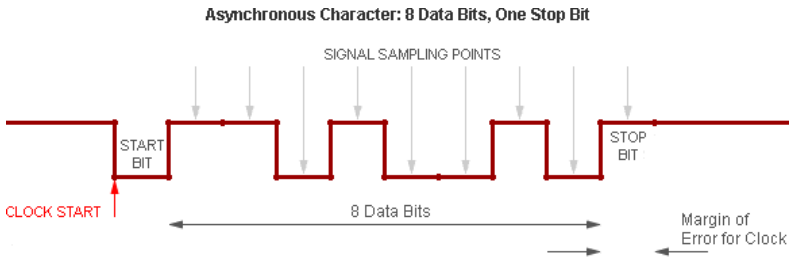




# Physical layer

- Wired
  - Parallel or Serial
    - Synchronization,
  - Serial,
    - Asynchronous
      - RS232C, RS422, RS485.
    - Synchronous
- Optical
- Wireless;
  - ZigBee, Bluetooth, Ethernet, Wireless HART,
  - Security; SSID, WPA (Wifi Protection Access)

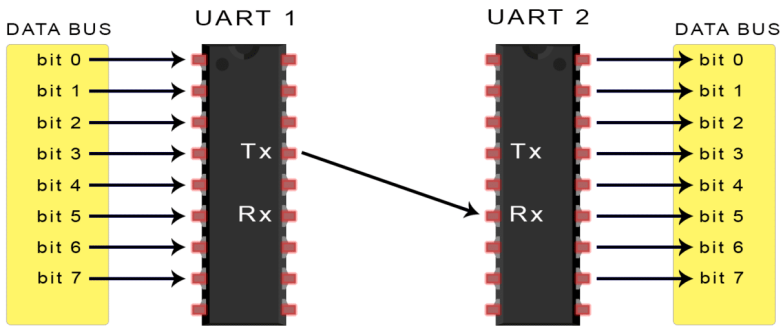
# Asynchronous serial communication (RS-232C / RS-422 / RS-485)

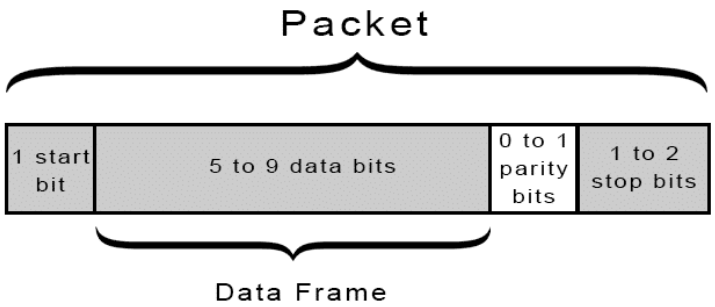



- Communication parameters
  - Baud rate (transmission speed)
    - 300, 600, .. ,9600, 19200, 38400, .. , 115200
  - Data bits (number of data bits)
    - 7 or 8
  - Parity (Error detection)
    - N:None, O:Odd, E:Even
  - Stop bits (number of stop bits)
    - 1, 1.5, 2

# Asynchronous serial communication (RS-232C / RS-422 / RS-485)




- UART
  - universal asynchronous receiver-transmitter,
  - FIFO buffer in receiver,
    - First in first out,
    - 16 characters,
  - Parity checking and error,
  - Framing error.

# C# programming

```
while (iOffset < iCntMax)
{
    iLen = rSerialPort.Read(bMsgBuf, iOffset, (bMsgBuf.Length - iOffset));
    iOffset += iLen;
}
for (iMsgCnt = 0; iMsgCnt
{
    sMsgBuf = sMsgBuf + (
}

ch (TimeoutException)

// Do nothing
```

Solution

tie

int SerialPort.Read(byte[] buffer, int offset, int count) (+ 1 overload)

Reads a number of bytes from the SerialPort input buffer and writes those bytes into a byte array at the specified offset.

Exceptions:
ArgumentNullException
InvalidOperationException
ArgumentOutOfRangeException
ArgumentException
TimeoutException

- Read and write to the port,
  - Read will normally not finish unless any characters are received,
  - Set the timeout parameters to get a timeout exception.
- Close the port when finished!

# Select serial port – list available serial ports

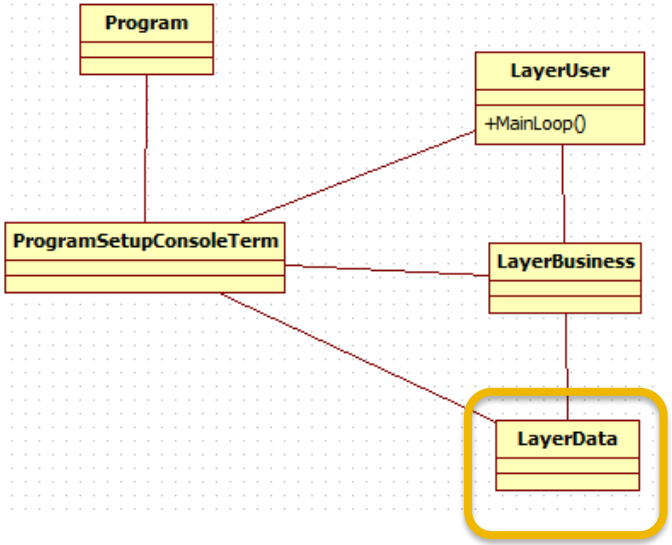
- using System.IO.Ports;
- string[] saComPortsNames;
- try

```
{
    saComPortsNames =
                SerialPort.GetPortNames();
}
catch
{
    saComPortsNames = null;
}
```

C:\Users\Nils-Olav\usr\HiT\FoU\Software\SerialPorts\NosCo...

```
---- Console Terminal : Main Menu ----
Current COM port:  None
-------------------------------------
N: Select a COM port
O: Open a selected COM port
T: Input the loopback test string
R: Raw dumping of COM port data
L: Test loopback of COM port data
G: GPS data from NMEA protocol (S:Start/Stop)
B: OBD protocol (S:Start/Stop)
Q: Quit console terminal

Select a function:n
Number of available COM ports =
1: COM5
2: COM1
3: COM9
Select id for COM port:
```

# Open()




```
public bool Open(string sPortName, out string sOpenMsg)
{
        if (bOpenPort == true)
                Close();
        try
        {
                rSerialPort = new SerialPort(sPortName, 4800, Parity.None, 8, StopBits.One);
                rSerialPort.ReadTimeout = 500;
                rSerialPort.Open();
                sOpenMsg = "Open <" + sPortName + "> serial port OK!";
                bOpenPort = true;
        }
        catch (Exception e)
        {
                sOpenMsg = "Error open <" + sPortName + "> serial port: " + e.Message;
                bOpenPort = false;
        }
        return bOpenPort;
}
```

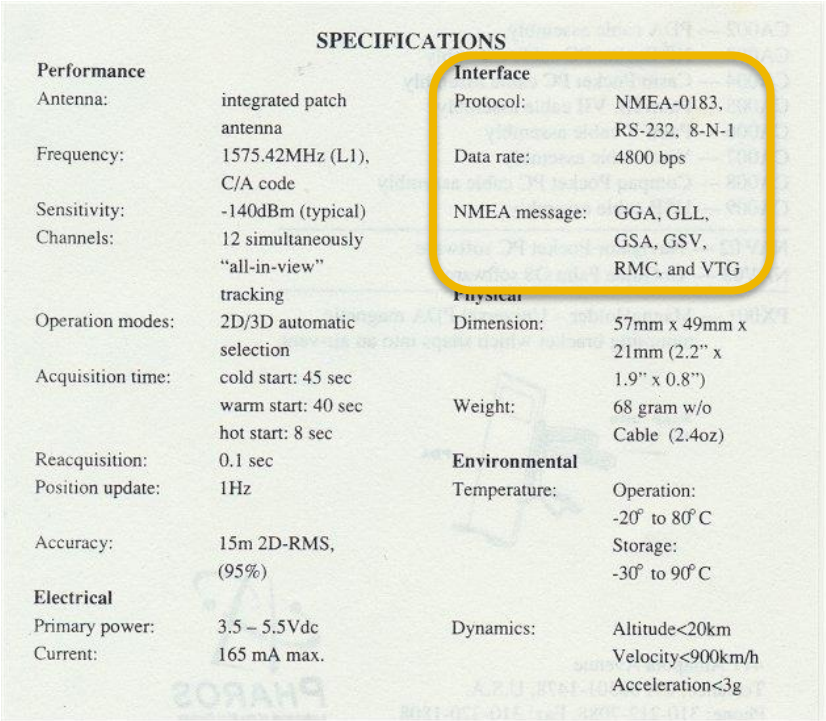
# Read()

```
public string Read(int iCntMax, bool bTimeoutMsg)
{
    int iLen = 0, iMsgCnt, iOffset;
    string sMsgBuf = “”;
    byte[] bMsgBuf;
     try
     {
         try { // Read function ➔ }
         catch (Exception e)
         {
             sMsgBuf = sMsgBuf + "<Exc=" + e.Message + ">";
         }
    }
    catch (Exception e)
    {
        sMsgBuf = sMsgBuf + "<PortErr=" + e.Message + ">";
    }
    return sMsgBuf;
}
```

```
bMsgBuf = new byte[iCntMax + 64];
iOffset = 0;
try
{
    while (iOffset < iCntMax)
    {
        iOffset += rSerialPort.Read(bMsgBuf, iOffset, (bMsgBuf.Length - iOffset));
     }
     for (iMsgCnt = 0; iMsgCnt < iOffset; iMsgCnt++)
     {
         sMsgBuf = sMsgBuf + Convert.ToChar(bMsgBuf[iMsgCnt]);
     }
}
catch (TimeoutException)
{
     if (iOffset > 0)
     {
         for (iMsgCnt = 0; iMsgCnt < iOffset; iMsgCnt++)
         {
             sMsgBuf = sMsgBuf + Convert.ToChar(bMsgBuf[iMsgCnt]);
         }
     }
}
```

# Communication protocols

- Loopback testing;
  - Interconnect TxD and RxD pins.
- GPS protocol;
  - NMEA protocol,
  - Transmitting cyclic messages.
- OBD protocol
  - Send request,
  - Wait for answer,
    - About 200 mSec.



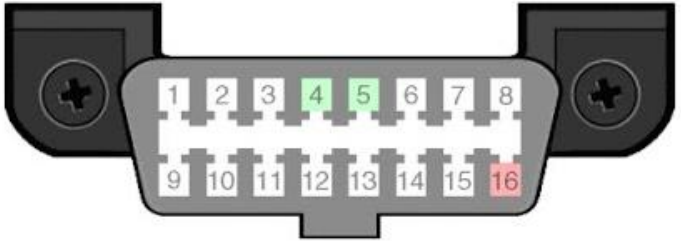
SPECIFICATIONS

| | | | |
|---|---|---|---|
| **Performance** | | **Interface** | |
| Antenna: | integrated patch antenna | Protocol: | NMEA-0183, RS-232, 8-N-1 |
| Frequency: | 1575.42MHz (L1), C/A code | Data rate: | 4800 bps |
| Sensitivity: | -140dBm (typical) | NMEA message: | GGA, GLL, GSA, GSV, RMC, and VTG |
| Channels: | 12 simultaneously "all-in-view" tracking | **Physical** | |
| Operation modes: | 2D/3D automatic selection | Dimension: | 57mm x 49mm x 21mm (2.2" x 1.9" x 0.8") |
| Acquisition time: | cold start: 45 sec<br>warm start: 40 sec<br>hot start: 8 sec | Weight: | 68 gram w/o Cable (2.4oz) |
| Reacquisition: | 0.1 sec | **Environmental** | |
| Position update: | 1Hz | Temperature: | Operation: -20° to 80° C<br>Storage: -30° to 90° C |
| Accuracy: | 15m 2D-RMS, (95%) | | |
| **Electrical** | | | |
| Primary power: | 3.5 – 5.5Vdc | Dynamics: | Altitude<20km<br>Velocity<900km/h<br>Acceleration<3g |
| Current: | 165 mA max. | | |

# OBD protocol

1. Setup serial port;
   1. 9600 Baud, 8 data bits, Parity None, 1 Stop bit.
2. Reset port:
   1. Send “ATZ”,
   2. Response is ELM327 version: ELM327v1.5<CR>
3. Select OBD protocol automatically,
   1. Send “ATSP 0”,
   2. Response is OK
4. Get ODB protocol (if wanted)
   1. Send “AT DP”,
   2. Response is AUTO,ISO 15765-4 (CAN11/500) (Example)
5. Get overview of PID support (optionally)
   1. Send 01 00
   2. Wait minimum 3 seconds
   3. Response is 41 00 + four bytes with active bits for active commands.
6. Standard commands: OK response is service code + 40

Data Link Connector (vehicle OBDII port)




1 Make/Model Specific
2 SAE J1850-PWM POS(+) or SAE J1850-VPW POS(+)
3 Make/Model Specific
4 Chassis Ground (all protocols)
5 Signal Ground (all protocols)
6 ISO15765-4 CAN-Bus High
7 ISO9141-2 K-Line or ISO14230-4 KWP2000 K-Line
8 Make/Model Specific
9 Make/Model Specific
10 SAE J1850-PWM NEG(-)
11 Make/Model Specific
12 Make/Model Specific
13 Make/Model Specific
14 ISO15765-4 CAN-Bus Low
15 ISO9141-2 L-Line or ISO14230-4 KWP2000 L-Line
16 +12v (always on) (all protocols)

# OBD protocol – standard commands

- Service value;
  - 01
- Request type;
  - 01 + PID
  - Speed: 010D
- Response;
  - 41 + PID + Value
- Standard PIDs;
- Special PIDs;

| PID (hex) | PID (Dec) | Data bytes returned | Description | Min value | Max value | Units | Formula[a] |
|---|---|---|---|---|---|---|---|
| 00 | 0 | 4 | PIDs supported [01 - 20] | | | | Bit encoded [A7..D0] == [PID $01..PID $20] See below |
| 01 | 1 | 4 | Monitor status since DTCs cleared. (Includes malfunction indicator lamp (MIL) status and number of DTCs.) | | | | Bit encoded. See below |
| 02 | 2 | 2 | Freeze DTC | | | | |
| 03 | 3 | 2 | Fuel system status | | | | Bit encoded. See below |
| 04 | 4 | 1 | Calculated engine load | 0 | 100 | % | $\frac{100}{255}A$ (or $\frac{A}{2.55}$) |
| 05 | 5 | 1 | Engine coolant temperature | -40 | 215 | °C | $A - 40$ |
| 06 | 6 | 1 | Short term fuel trim—Bank 1 | -100 (Reduce Fuel: Too Rich) | 99.2 (Add Fuel: Too Lean) | % | $\frac{100}{128}A - 100$ (or $\frac{A}{1.28} - 100$) |
| 07 | 7 | 1 | Long term fuel trim—Bank 1 | | | | |
| 08 | 8 | 1 | Short term fuel trim—Bank 2 | | | | |
| 09 | 9 | 1 | Long term fuel trim—Bank 2 | | | | |
| 0A | 10 | 1 | Fuel pressure (gauge pressure) | 0 | 765 | kPa | $3A$ |
| 0B | 11 | 1 | Intake manifold absolute pressure | 0 | 255 | kPa | $A$ |
| 0C | 12 | 2 | Engine RPM | 0 | 16,383.75 | rpm | $\frac{256A + B}{4}$ |
| 0D | 13 | 1 | Vehicle speed | 0 | 255 | km/h | $A$ |
| 0E | 14 | 1 | Timing advance | -64 | 63.5 | ° before TDC | $\frac{A}{2} - 64$ |
| 0F | 15 | 1 | Intake air temperature | -40 | 215 | °C | $A - 40$ |
| 10 | 16 | 2 | MAF air flow rate | 0 | 655.35 | grams/sec | $\frac{256A + B}{100}$ |
| 11 | 17 | 1 | Throttle position | 0 | 100 | % | $\frac{100}{255}A$ |
| 12 | 18 | 1 | Commanded secondary air status | | | | Bit encoded. See below |
| 13 | 19 | 1 | Oxygen sensors present (in 2 banks) | | | | [A0..A3] == Bank 1, Sensors 1-4. [A4..A7] == Bank 2... |
| | | | Oxygen Sensor 1 | | | | |

# Reading service 01 values: current data

- Start getting values
  - Send text buffer;
    - Service code + PID code + <CR>
  - Wait 500 mSec.
  - Read serial port
    - Status + PID code + answer + <CR>
  - Get next value, or first value
- Examples:
  - Send 010D<CR> : Answer 410D1C <CR>
    - PID: Speed = 0x0D
    - Speed: 1C = 28 km/h
  - Send 010C<CR>: Answer 410C541B<CR>
    - PID: RPM = 0x0C
    - RPM: A= 0x54 = 84 / B=0x1B = 27
    - RPM = ((A*256) +B) / 4 = 5382.75

| Tx | | Mode | Pid | <CR> | |
|---|---|---|---|---|---|
| Rx | | Status | Pid | Data | <CR> |

Status OK = Mode + 0x40

| | | | | | | |
|---|---|---|---|---|---|---|
| 0B | 11 | 1 | Intake manifol | 255 | kPa | $A$ |
| 0C | 12 | 2 | Engine RPM | 16,383.75 | rpm | $\frac{256A+B}{4}$ |
| 0D | 13 | 1 | Vehicle speed | 255 | km/h | $A$ |
| | | | | | ° before | $A$ |

# OBD system (basic functionality)

In the car:

- OBD only working with ignition on,
- No connection with the cloud system,
- Hands free solution,
  - No input if speed > 5 km/h ?
- Init ELM27 device,
- Start reading a set of parameters,
- Save values on CVS file,
  - Date and Time, Value #1, Value #2, .. <CR>
- Go back reading and updating the parameters.

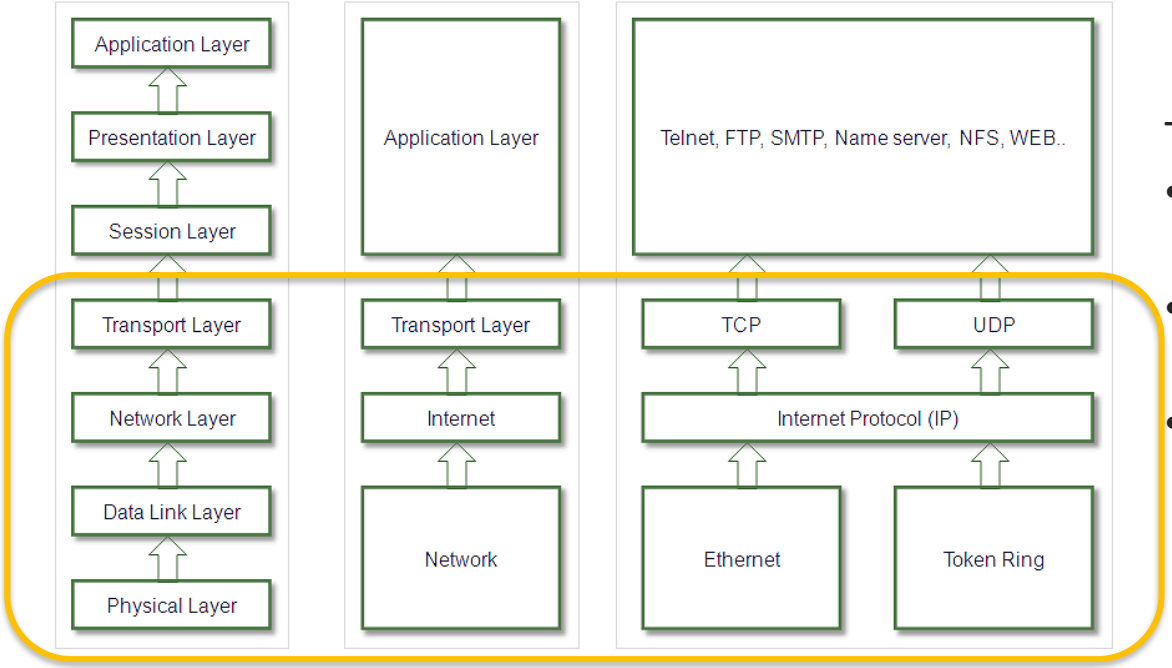
After driving the car:

- Upload the CVS file into the system,
- Update database/cloud system,
- Update new driving info,
  - Car usage,
  - Fuel cost,
  - Service cost,
  - MOT (PKK) information,
  - ….

# Serial port demo

- Loopback test;
  - Testing of local hardware and software.
- NMEA / GPS data;
  - Any data inside?
- Temp / Light data from an Arduino device;
  - Sending data every 5th second.
- OBDII data?

# Transport layer

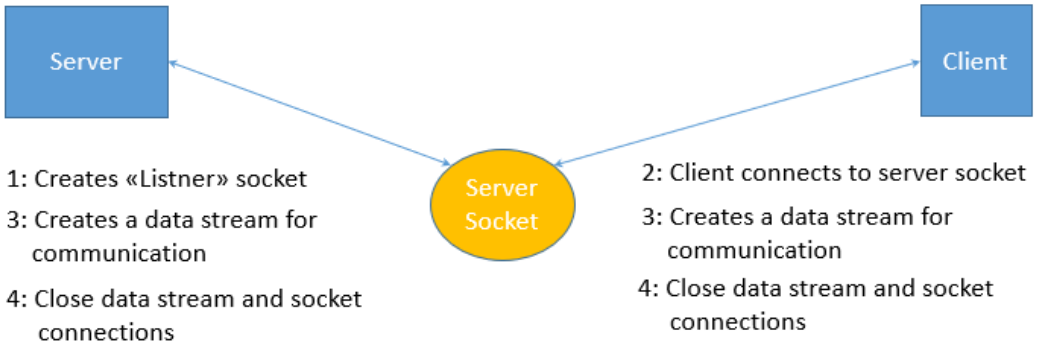



TCP/IP using several parameters:

- TCP/IP address;
  - IP v4 or IP v6,
- Protocol type;
  - TCP or UDP,
- Port number;
  - «Reserved» 0 - 1023
  - «Free» 1024 - …

# TCP/IP communication

- Using a socket,
- Based on server / client,
  - Minimum two applications,
    - Server and minimum one client,
  - Communication media,
    - Owns by the server,
    - Server makes «Listner» socket,
  - Client(s) connect.
    - To server «Listner» socket,
    - TCP/IP address,
  - Same protocol type and port number.



# C# programming (server)

```
Server: Available IP addresses for the server node:
 IP adrress[1] = fe80::24fb:a678:c327:aaaa%9
 IP adrress[2] = 2a01:799:ae0:800:70fe:ab1:31c:6bef
 IP adrress[3] = 2a01:799:ae0:800:24fb:a678:c327:aaaa
 IP adrress[4] = 192.168.13.195
```

- Get the IP address of the node:

```
ipHostInfo = Dns.GetHostEntry(Dns.GetHostName());
ipAddress = ipHostInfo.AddressList[0];
```

- Server setup:

```
tcpListener = new TcpListener(ipAddress, iSocketPortId);
tcpListener.Start(5);
```

- Create a socket connection between the server and client:

```
tcpClient = tcpListener.AcceptTcpClient();
tcpNetworkStream = tcpClient.GetStream();
```

- Close connection after the communication:  (tcpNetworkStream and tcpClinet)

# C# programming (client)

- Get the IP address of the node: (must adjust to the remote node)

```
ipHostInfo = Dns.GetHostEntry(Dns.GetHostName());
ipAddress = ipHostInfo.AddressList[0];
```

- Connect to the server:

```
tcpClient = new TcpClient(ipAddress.ToString(), iSocketPortId);
```

- Connect to the socket connection:

```
tcpNetworkStream = tcpClient.GetStream();
```

- Close connection after the communication:  (tcpNetworkStream and tcpClinet)

# C# programming (read and write)

- The socket stream will be used for reading and writing bytes,
  - Need some sort of protocol to understand the contents,
  - TCP/IP is NOT defining any way of coding the information.
- Reading:

```
byte[] baBuffer = new byte[64];
int iLen = tcpNetworkStream.Read(baBuffer);
String sBuffer = Encoding.ASCII.GetString(baBuffer);
sBuffer = sBuffer.TrimEnd('\0');
```

- Writing:

```
string sBuffer = DateTime.Now.ToString() + ": Server command=<" + sBuffer + ">";
byte[] baBuffer = Encoding.ASCII.GetBytes(sBuffer);
tcpNetworkStream.Write(baBuffer);
```

# C# programming – console application

- Start the server as a thread,
- Two commands:
  - LIST: IP address of the server
  - QUIT: stop the server and client
- Source code in the compendium.

- Extension:
  - Two applications?
  - More commands?

- Graphical CHAT application