Features

- Compatible with FIAT Motors
- RS232 Communication at 9600 Baud
- All Diagnostic modes supported
- K-Line diagnostic
- Support Airbag and ABS K-Lines standard
- Listening possible C-CAN and B-CAN in v2.0
- Read clear manufacturer DTCs
- Partially hardware compatible to 2600

ÖZEN ELEKTRONIK

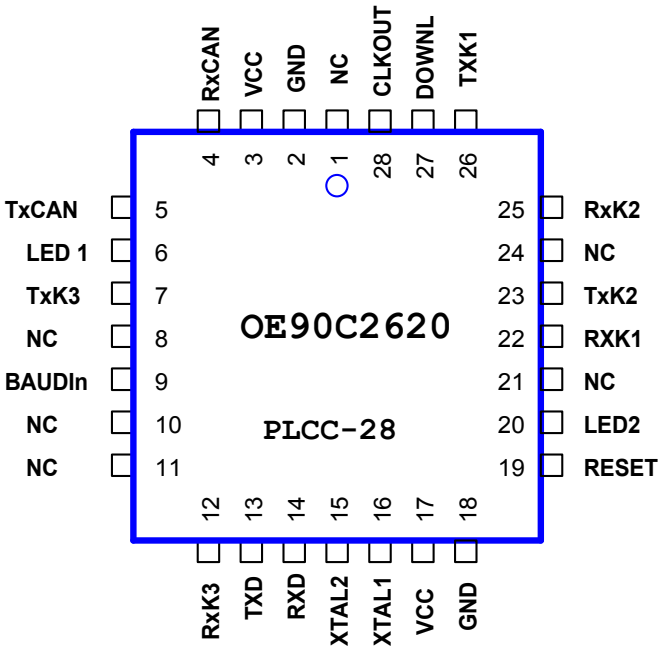
FIAT to RS232 gateway

OE90C2620

This chip is partially compatible to 2600.
İf fitted on a 2600 board this chip will works on K-line diagnosis (pin 7) and it can handle High speed CAN Messaging.

Description

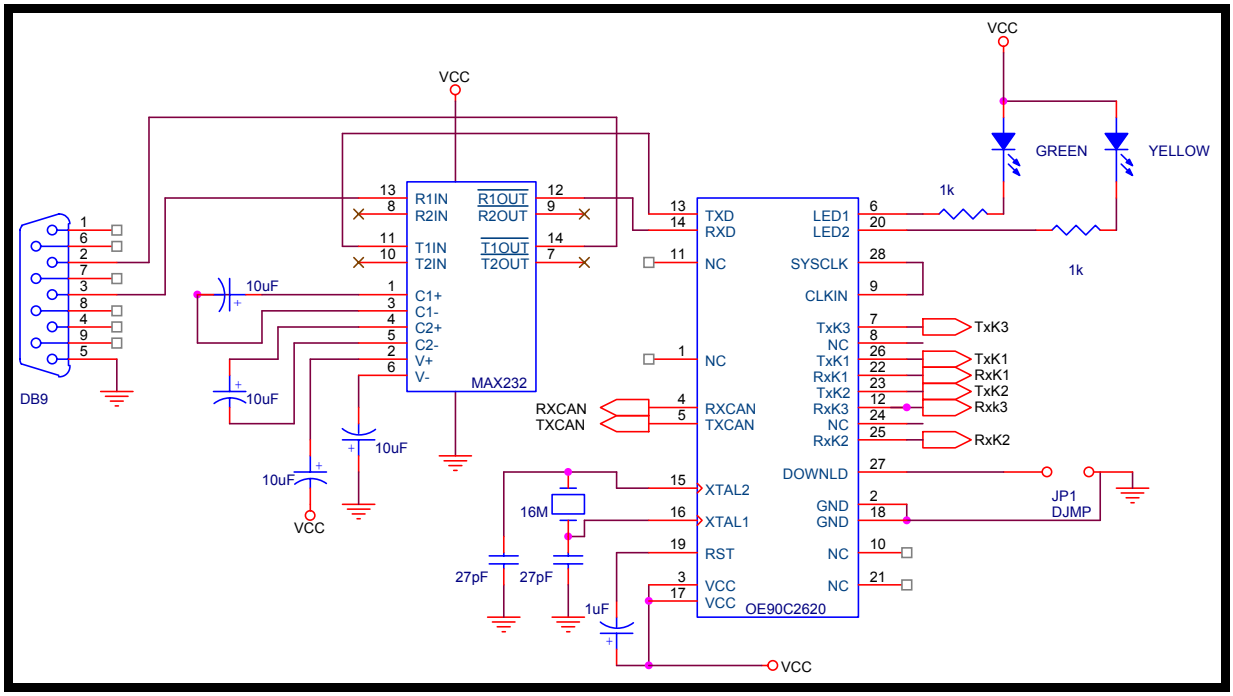
OE90C2620 is specially designed for FIAT motor company cars. This chip allows to build a complete diagnostic system using a PC or directly in small device with LCD display. Some other brand like Alfa and Lancia can be tested. All diagnostic mode are implemented . Some commands are implemented to allow direct access to different ECUs. This chip is partially plug play compatible with mobydic2600 Hardware and PCB.



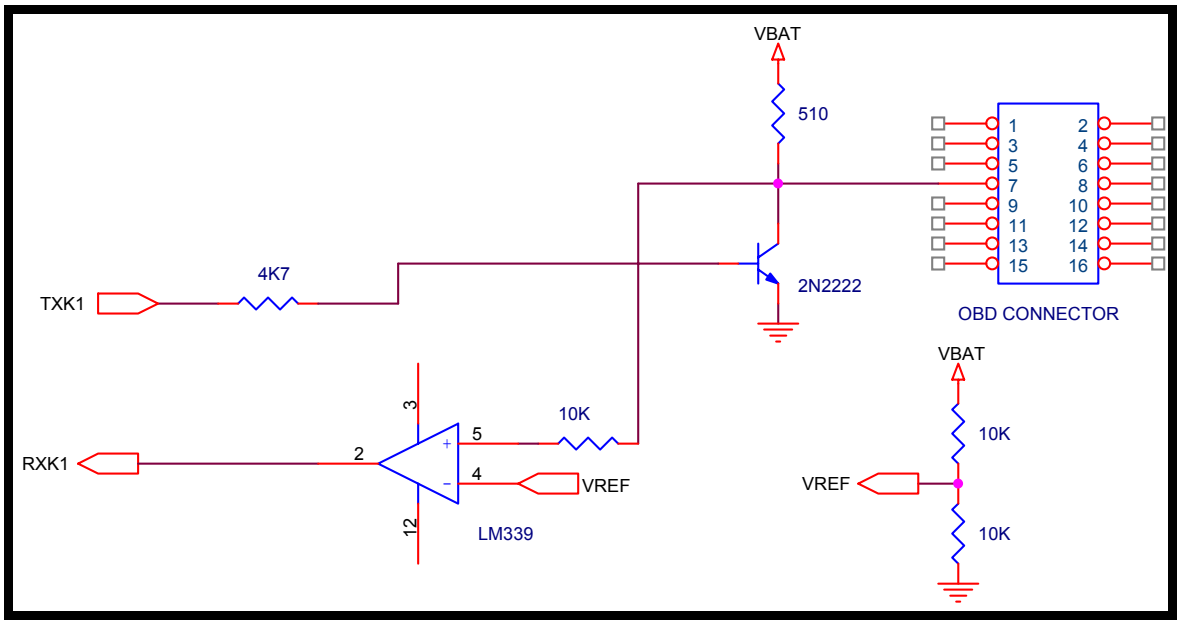
Pin description

Pin	Pin Name	Type	Description
1	NC		
2	GND		Ground
3	VCC		Supply voltage
4	RXCAN	I	B-CAN BUS input
5	TXCAN	O	B-CAN BUS output
6	LED1	O	Connected (Main ECU based) LED max 5 mA for low current LED
7	TxK3		Airbag K-line output
8	NC		
9	BAUDIN		RS232 Baudrate input clock
10	NC		
11	NC		
12	RxK3		Airbag K-line input
13	TXD	O	RS232 output
14	RXD	I	RS232 input
15	XTAL2	I	16 Mhz crystal input
16	XTAL1	I	16 MHz crystal input
17	VCC	I	Supply voltage
18	GND	I	Ground
19	RESET	I/O	A high level on this pin during 2 machine cycles while the oscillator is running resets the device.
20	LED2	O	LED output to indicate the frames exchange
21	NC		
22	RXK1	I	K-line diagnostic input
23	Txk2	I	K-line output to ABS
24	NC		
25	RXK2	O	K-Line input from ABS
26	TxK1	O	K-Line diagnostic output
27	DOWNLD	I	A low on this pin puts the device in download mode
28	CLKOUT	O	Clock output for RS232 baud rate in

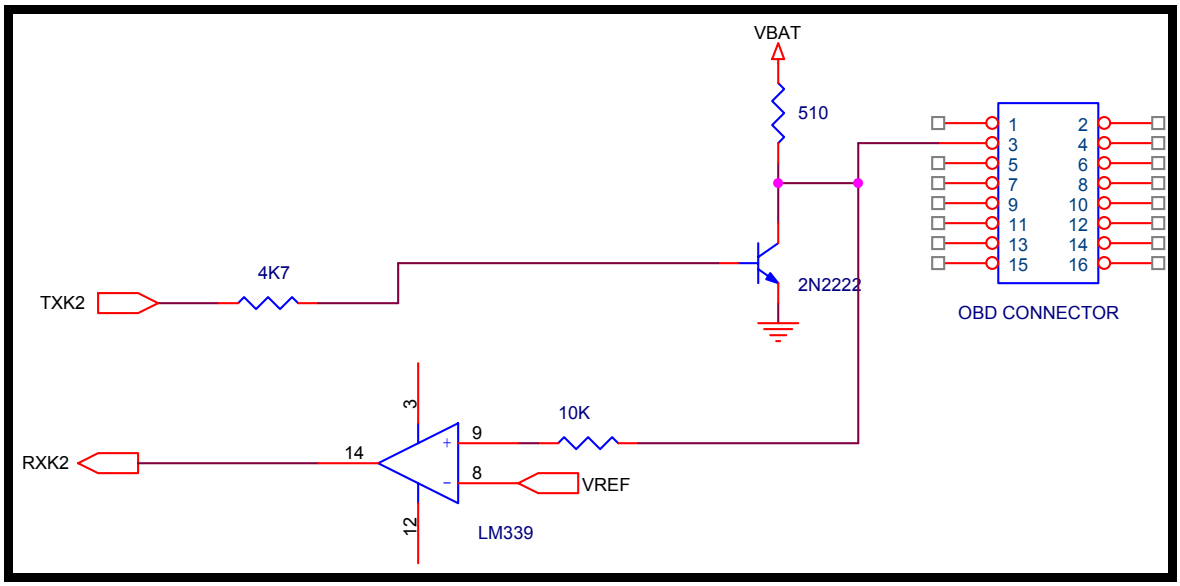
Hardware application schematic



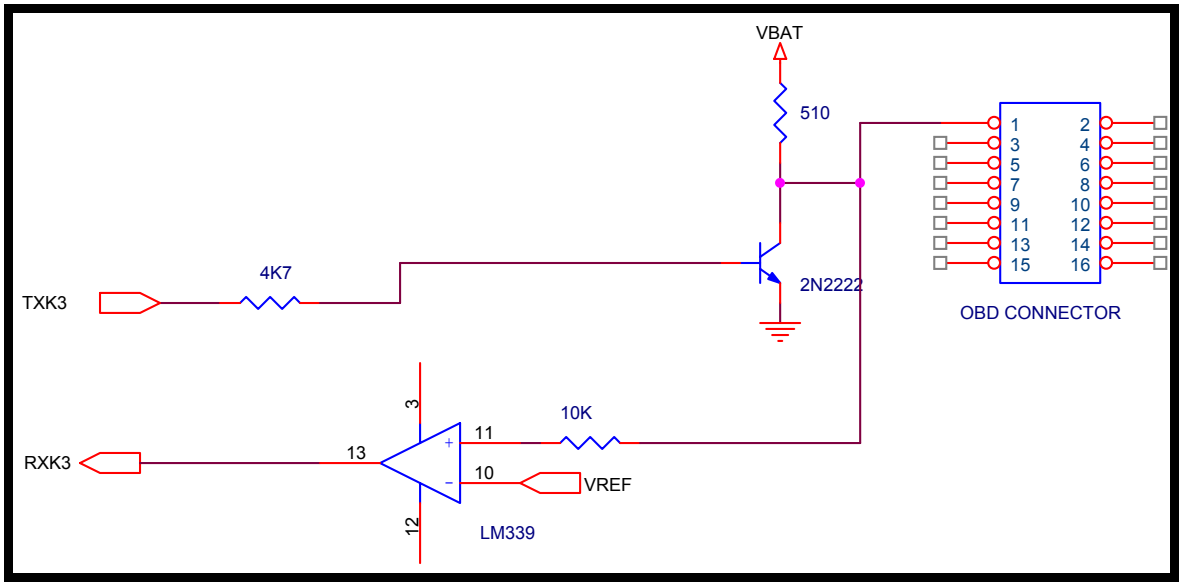
Typical connection of OE90C2620



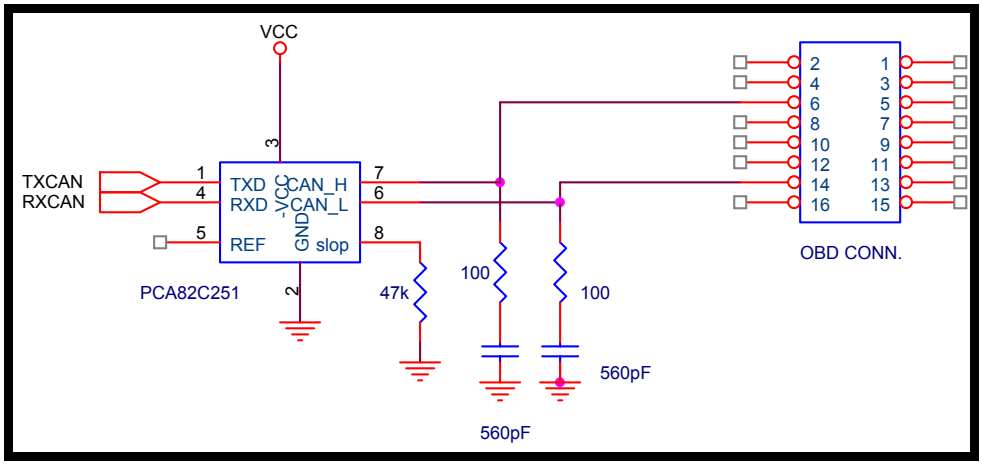
Typical connection of K-Line



Typical connection of Airbag K-Line



Typical connection of ABS K-Line



Typical connection of CAN

FIAT SPECIFICATIONS

K-Line diagnosis on pin 7

The implementation of fiat KWP2000 protocol is based on fiat document 07223. The structure of the messages is desrcibed in the document ISO14230-2. The ECU accepts both 5 baud and Fast initialization. The baud rate used by both the ECU and Tester is 10.4 Kb. mOByDic Interbyte time is fixed at 8 mS , can be adjusted between 5..20 mS. MOByDic uses only the fast initialization. The Keywords are KW2-KW1 = 8FEF.

The purpose of this specification is to allow access to current data values, including analogue inputs and outputs, digital inputs and outputs, and system status information. The request for information includes a local identification (LI) value that indicates to the on-board system the specific information requested. LI specifications, scaling information, and display formats are engine specific. The ECU will respond to this message by transmitting the requested data value last determined by the system. All data values returned for sensor readings will be actual readings, not default or substitute values used by the system because of a fault with that sensor.

The request/Response message format :

10LL LLLL	ECU_ADDRESS	F1	data bytes	CRC
10LL LLLL	F1	ECU_ADDRESS	data bytes	CRC

LLLLLL : length of the Data bytes

The Request and response Data bytes presentation format :

Data Byte	Parameter Name	Hex Value
#1		
#2		
#n		

CAN BUS STRUCTURE

The standard frame structure is based ob CAN 2.0B Active protocol specification with 11 bit ID and 500kB / 50 KB baudrate.
The message identifier is assigned with Class , Category and node address. The 8 bytes data field is interpreted as 64 bits (0..63) and any signal is mapped in this field. The Data mapping is defined in FIAT 7325 and any Diagnostic request response is defined in FIAT 7274 specification.

.

ABS K-Line diagnostic

The implementation of fiat ABS - KWP2000 protocol is based on fiat document 2022 . The structure of the messages is desrcibed in the document ISO14230-2. The ECU accepts only fast initialization. The baud rate used by both the ECU and Tester is 10.4 Kb. mOByDic Interbyte time is fixed at 8 mS , can be adjusted between 5..20 mS. MOByDic uses only the fast initialization.

The purpose of this specification is to allow access to current ABS data values, including analogue inputs and outputs, digital inputs and outputs, and system status information. The request for information includes a local identification (LI) value that indicates to the on-board system the specific information requested. LI specifications, scaling information, and display formats are engine specific. The ECU will respond to this message by transmitting the requested data value last determined by the system. All data values returned for sensor readings will be actual readings, not default or substitute values used by the system because of a fault with that sensor.

The request/Response message format :

10LL LLLL	ECU_ADDRESS	F1	data bytes	CRC
10LL LLLL	F1	ECU_ADDRESS	data bytes	CRC

LLLLLL : length of the Data bytes

The Request and response Data bytes presentation format :

Data Byte	Parameter Name	Hex Value
#1		
#2		
#n		

Airbag K-Line diagnostic

The implementation of fiat Airbag -protocol is based on fiat document 07279 . The structure of the messages is desrcibed in the document ISO9141 and FIAT 07234 . The ECU accepts 5 Baud initialization. The baud rate used by both the ECU and Tester is 4800 bd. Communication is based on Kw81 protocol.
For the flexibility , mobydic can handle it as KWP2000 too. Reading snapshoots is provided.

SOFTWARE INTERFACING TO CHIP

OE90C2620 communicates via RS232 with 9600/8/n/1 parameter . A Software Handshake is necessary prior to send a message because of auto-keepaliving. PC is master and mOByDic is slave. The communication works on half duplex basis. The 2620 has a communication timeout of 2 sec. For 2 bytes command without software handshaking the user shall wait 100 ms after sending 1.byte prior to send the second byte.

RS232 device driver routines

İf the user sends a 1 to mOByDic , mOByDic responds with 1 . This function allows a line checking or device checking. Some delphi exemples show how to begin to communicate with mOByDic. Please note that the following routines dont intend to be used in a professional application , they only allow the user to understand the messaging strategy of mOByDic 2620.

```
unit mOByDic_device_driver;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls;

var

  hCommFile     : THandle;
  DCB           : TDCB;                    // structure for rs232
  read_Error    : boolean;                 // RS232 Read error by timeout
  RS232_Buffer  : array [0..255] of byte;  // buffer for incoming bytes
```

```
implementation

//***************************************************************************
```

```
//*                                                                            *
//* Function: open and initialize COM1 port  9600 baud / 8 / n / 1             *
//* Input   : ---------                                                        *
//* Output  :                                                                  *
//*                                                                            *
//******************************************************************************

Procedure Open_COMM;
  begin

      // change COM1 if another port will be used

    hCommFile := CreateFile('COM1' , GENERIC_READ or GENERIC_WRITE,
                            0 , Nil, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, 0);


    GetCommState(hCommFile,DCB);    // set comm parameter
    with DCB do
      begin
        Baudrate:= CBR_9600;
        ByteSize:= 8;
        Parity  := NOPARITY;
      end;
    SetCommState(hCommFile,DCB);


    if hCommFile = INVALID_HANDLE_VALUE then  // error at open port
      begin
        ShowMessage ('ERROR OPENING PORT');
        exit;
      end;

  end;

//******************************************************************************
//*                                                                            *
//* Function: close a open comm port                                           *
//* Input   :                                                                  *
//* Output  :                                                                  *
//*                                                                            *
//******************************************************************************

Procedure close_COMM;
  begin
    closeHandle(hCommFile);
  end;

//******************************************************************************
//*                                                                            *
//* Function: clearBuffers, clear rs232 Tx and Rx buffers                      *
//* Input   :                                                                  *
//* Output  :                                                                  *
//*                                                                            *
//******************************************************************************

  Procedure clearBuffers;
    begin
      purgecomm(hcommfile,PURGE_TXCLEAR);
      purgecomm(hcommfile,PURGE_RXCLEAR);
    end;




//******************************************************************************
//*                                                                            *
//* Function: write a byte to open RS232 port no timeout occurs                *
//* Input   : tosend : byte to send                                            *
//* Output  :                                                                  *
//*                                                                            *
//******************************************************************************
```

```
Procedure write_RS232 ( tosend:byte);
  Var sent:dword;
  begin
   WriteFile (hCommFile,tosend,1,sent,nil);
  end;


//*****************************************************************************
//*                                                                           *
//* Function: Read a byte from RS232                                          *
//* Input   : Timeout value in mS to wait on first byte                       *
//* Output  : received char  , Read_error flag is true if any error           *
//*                                                                           *
//*****************************************************************************

Function read_RS232(Timeout_value:word) : byte;
  var
    TimeOutBuffer:PCOMMTIMEOUTS;
    bytesread:dword;
    read_byte:byte;

  begin

    //timeout parameter setting

    GetMem(TimeoutBuffer,sizeof(COMMTIMEOUTS));
    GetCommTimeouts (hCommFile,TimeoutBuffer^);
    TimeoutBuffer.ReadTotalTimeoutMultiplier:=0;
    TimeoutBuffer.ReadTotalTimeoutConstant:=timeout_Value;
    SetCommTimeouts (hCommFile,TimeoutBuffer^);

    ReadFile(hCommFile,Read_byte,1,bytesread,nil);
    if (BytesRead = 0)
          then begin
            Read_RS232:= 0;
            read_error := true;
          end
          else begin
            read_error := false;
            Read_RS232:= read_byte;
          end;
  end;

end.
```

MOByDic 2620 Command list

Commande	01
Fonction	First contact to 2620 / line checking
Request	01
Pos. Response	01
Neg. Response	N/A

Exemple for command 01

```
Open_COMM;
clearbuffers;
write_rs232(1);                        // send first contact message
rs232_buffer[0]:=read_rs232(100);      // get the response with 100 ms timeout
  if read_error
     then ........                     // no contact
     else if rs232_buffer[0]=1
            then ........              // first contact OK device existes
            else ........              //  no contact
```

Commande	03
Fonction	Read serial number of 2620
Request	03
Pos. Response	03 HB LB HB = high byte of serial number LB = low byte of serial number Serial# = (256*HB) + LB ***Actually not used and give 00 00 back***
Neg. Response	N/A

Exemple for command 03

```
Open_COMM;
clearbuffers;
write_rs232(3);                        // send read serial number command
rs232_buffer[0]:=read_rs232(100);      // get the response with 100 ms timeout
if read_error then exit                // no response
rs232_buffer[1]:=read_rs232(10);       // get the response with 10 ms timeout
if read_error then exit                // no response
rs232_buffer[2]:=read_rs232(10);       // get the response with 10 ms timeout
if read_error then exit                // no response
Serial:=(256*rs232_buffer[1])+rs232_buffer[2]; // get serial
```

Commande	04
Fonction	Read version of 2620
Request	04
Pos. Response	04 HB LB HB = high byte version LB = low byte of version Version# = (256*HB) + LB (220 decimal => ver 2.20)
Neg. Response	N/A

Exemple for command 04

```
Open_COMM;
clearbuffers;
write_rs232(4);                      // send read version number command
rs232_buffer[0]:=read_rs232(100);    // get the response with 100 ms timeout
if read_error then exit              // no response
rs232_buffer[1]:=read_rs232(10);     // get the response with 10 ms timeout
if read_error then exit              // no response
rs232_buffer[2]:=read_rs232(10);     // get the response with 10 ms timeout
if read_error then exit              // no response
Version:=(256*rs232_buffer[1])+rs232_buffer[2]; // get version in x100 format
```

Commande	05
Fonction	Soft Reset of 2620
Request	05
Pos. Response	ACK (06)
Neg. Response	N/A

Exemple for command 05

```
Open_COMM;
clearbuffers;
write_rs232(5);                      // send soft reset command
rs232_buffer[0]:=read_rs232(100);    // get the response with 100 ms timeout
if read_error then exit              // no response
.................
```

Commande	06
Fonction	Read chip ident of mOByDic
Request	06
Pos. Response	06 HB LB HB = high byte of chip ident LB = low byte of chip ident Version# = (256*HB) + LB 2620 = OE90C2620
Neg. Response	N/A

Exemple for command 06

```
Open_COMM;
clearbuffers;
write_rs232(6);                       // send read chip ident command
rs232_buffer[0]:=read_rs232(100);     // get the response with 100 ms timeout
if read_error then exit               // no response
rs232_buffer[1]:=read_rs232(10);      // get the response with 10 ms timeout
if read_error then exit               // no response
rs232_buffer[2]:=read_rs232(10);      // get the response with 10 ms timeout
if read_error then exit               // no response
chip_ident:=(256*rs232_buffer[1])+rs232_buffer[2]; // get chip ident
```

Commande	10
Fonction	Set ECU Address on FIAT KWP2000 / pin 7
Remarque	Default setting is $10 (PCM)
Request	10 , ECU_Address
Pos. Response	ACK (06)
Neg. Response	N/A

Exemple command 10 how setting the header in mObydic to EOBD via Diagnosis line

```
Open_COMM;
clearbuffers;
write_rs232(10);                      // send set address command
write_rs232($33);                     // send EOBD address to mOByDic
rs232_buffer[0]:=read_rs232(100);     // get the response with 100 ms timeout
if read_error then exit               // no response
if rs232_buffer[0] = ACK
    then ..................           // handle
```

Commande	11
Fonction	Connect to ECU
Remarque	This command allows a immediat connection to ECU with KWP2000 fast init. This command uses default ECU address or address set by command 10. An Auto keepalive function is actived sending the tester present message to ECU. Setting the ECU Address to EOBD provides a keep aliving by sending service 1 PID 0 request message . The StartComm String is automatically generated by the chip.
Request	11
Pos. Response	11, Length , <KWP2000 response >
Neg. Response	NACK not connected please turn ignition key on

Exemple command 11 how connecting to PCM via Diagnosis line

```
Open_COMM;
clearbuffers;
write_rs232(11);                      // send connect command
rs232_buffer[0]:=read_rs232(100);     // get the response with 100 ms timeout
if read_error then exit               // no response


length:=read_rs232(10);                // response message length
for i:=1 to rs232_length do
        begin
         rs232_buffer[i]:=read_rs232(10); // get data
         if read_error then exit          // no response
        end;
```

Commande	12
Fonction	Disconnect from Engine ECU
Remarque	This command allows a immediate disconnection from ECU . This command stops only the keep aliving . The user needs to wait 5 sec. till he begins with a new initializing
Request	12
Pos. Response	ACK (indicates that the keepaliving is stopped)
Neg. Response	NACK ECU was not connected

Exemple command 12 how disconnect from ECU via diagnosis line

```
clearbuffers;
write_rs232(12);                      // send disconnect command
rs232_buffer[0]:=read_rs232(100);     // get the response with 100 ms timeout
if read_error then exit               // no response
if rs232_buffer[0] = ACK
    then sleep(5000)                  // wait for disconnecting
    else                              // was not connected !!!
```

Commande	13
Fonction	Communicate with ECU
Remarque	
Request	Send 13 to mobydic Wait for 06 (ACK) after receiving ACK user has only 2 sec. to continue Send length , data see exemple length max 60 bytes
Pos. Response	13 , length , data length=0 indicating no data
Neg. Response	NACK ECU not connect please first connect

This command is used to communicate with connected ECU in KWP2000 format.

A Normal KWP2000 Message Format

Header Bytes			Data Bytes							checksum
Length	Target	Source	1	2	3	4	5	6	7	CKS / CRC
80+length	ECU address	F1								CKS

Requested format for mobydic2620

Length	Data Bytes						
	1	2	3	4	5	6	7

Note that Checksum is automatically calculated by modydic2620

Mobydic responds with

Header Bytes			Data Bytes							checksum
Length	Target	Source	1	2	3	4	5	6	7	CKS / CRC
80+length	F1	ECU addr.								CKS

Exemple 1 get RPM using RLI

Length	Data Bytes						
	1	2	3	4	5	6	7
2	21	30					

Mobydic responds after 13 , length with :

Header Bytes			Data Bytes							checksum
Length	Target	Source	1	2	3	4	5	6	7	CKS / CRC
85	F1	ECU Addr.	62	30	xx	xx				CKS

Xx,xx = RPM data

Delphi exemple for getting RPM using the RLI

```
clearbuffers;
write_rs232(13);                          // send communicate command
rs232_buffer[0]:=read_rs232(100);         // get the response with 100 ms timeout
if read_error then exit                   // no response
if rs232_buffer[0] = ACK
    then begin
      write_rs232(02);                     // 2 bytes follow
      write_rs232($21);                    // get data by RLI
      write_rs232($30);                    // RLI for RPM
      rs232_buffer[0]:=read_rs232(250); // response 13
      if read_error then exit              // no response
      rs232_buffer[1]:=read_rs232(10);  // response message length
      for i:=2 to rs232_buffer[1]+1 do
        begin
         rs232_buffer[i]:=read_rs232(10); // get data
         if read_error then exit           // no response
        end;
    end
    else                                   // surely a NACK check length!!!
```

After succesfully reception following data are in RS232_Buffer

Rs232_buffer	Value	Remarque
[0]	13	Command 13
[1]	8	String length
[2]	85	Kwp message header byte 1
[3]	F1	Kwp message header byte 2
[4]	ECU Addr.	Kwp message header byte 3
[5]	61	Response to 21 read by RLI
[6]	30	RLI for RPM
[7]	xx	RPM data
[8]	xx	RPM data
[9]	CKS	Checksumm of KWP2000 message

Exemple 2 Read DTCs by status from ECU

Send format for mobydic2620

Length	Data Bytes						
	1	2	3	4	5	6	7
4	18	00	FF	00			

Mobydic responds after 13 , length with :

Header Bytes			Data Bytes							checksum
Length	Target	Source	1	2	3	4	5	6	7	CKS / CRC
8x	F1	ECU Addr.	58	01	dd	dd	ss			CKS

```
clearbuffers;
write_rs232(13);                          // send communicate command
rs232_buffer[0]:=read_rs232(100);         // get the response with 100 ms timeout
```

```
if read_error then exit                  // no response
if rs232_buffer[0] = ACK
    then begin
      write_rs232 (04);                   // 4 bytes follow
      write_rs232($18);                   // get DTCs
      write_rs232($00);                   //
      write_rs232($ff);                   // for all group
      write_rs232($00);                   //
      rs232_buffer[0]:=read_rs232(250); // get response 13
      if read_error then exit             // no response
      rs232_buffer[1]:=read_rs232(10);  // response message length
      for i:=2 to rs232_buffer[1]+1 do
        begin
         rs232_buffer[i]:=read_rs232(10); // get data
         if read_error then exit           // no response
        end;
    end
    else .......                           // surely a NACK check length!!!
```

After succesfully reception following data are in RS232_Buffer

Rs232_buffer	Value	Remarque
[0]	13	Command 13
[1]	9	String length
[2]	8x	Kwp message header byte 1
[3]	F1	Kwp message header byte 2
[4]	ECU Addr.	Kwp message header byte 3
[5]	58	Response to 18 read DTC
[6]	1	Number of DTC
[7]	DTChb	DTC High byte
[8]	DTClb	DTC low byte
[9]	Statusof DTC	Status of DTC
[10]	CKS	Checksumm of KWP2000 message

Exemple 3 Clear DTCs

Send format for mobydic2620

Length	Data Bytes						
	1	2	3	4	5	6	7
3	14	FF	00				

Mobydic responds after 13 , length with :

Header Bytes			Data Bytes							checksum
Length	Target	Source	1	2	3	4	5	6	7	CKS / CRC
8x	F1	ECU Addr.	54	FF	00					CKS

```
clearbuffers;
write_rs232(13);                         // send communicate command
rs232_buffer[0]:=read_rs232(100);        // get the response with 100 ms timeout
if read_error then exit                  // no response
if rs232_buffer[0] = ACK
    then begin
```

```
      write_rs232 (03);                 // 3 bytes follow
      write_rs232($14);                 // get DTCs
      write_rs232($ff);                 // for all group
      write_rs232($00);                 //
      rs232_buffer[0]:=read_rs232(250); // get response 13
      if read_error then exit           // no response
      rs232_buffer[1]:=read_rs232(10);  // response message length
      for i:=2 to rs232_buffer[1]+1 do
        begin
         rs232_buffer[i]:=read_rs232(10); // get data
         if read_error then exit          // no response
        end;
    end
    else .......                         // surely a NACK check length!!!
```

After succesfully reception following data are in RS232_Buffer

Rs232_buffer	Value	Remarque
[0]	13	Command 13
[1]	7	String length
[2]	8x	Kwp message header byte 1
[3]	F1	Kwp message header byte 2
[4]	ECU Addr.	Kwp message header byte 3
[5]	54	Response to 14 clear DTC
[6]	FF	All group
[7]	00	
[8]	CKS	Checksumm of KWP2000 message

Exemple 3 EOBD Handling (read service 1 PID 0)

Send format for mobydic2620

Length	Data Bytes						
	1	2	3	4	5	6	7
2	01	00					

Mobydic responds after 13 , length with :

Header Bytes			Data Bytes							checksum
Length	Target	Source	1	2	3	4	5	6	7	CKS / CRC
8x	F1	ECU Addr.	41	00	pp	pp	pp	pp		CKS

```
clearbuffers;
write_rs232(13);                        // send communicate command
rs232_buffer[0]:=read_rs232(100);       // get the response with 100 ms timeout
if read_error then exit                 // no response
if rs232_buffer[0] = ACK
    then begin
      write_rs232 (03);                 // 3 bytes follow
      write_rs232($14);                 // get DTCs
```

```
      write_rs232($ff);                 // for all group
      write_rs232($00);                 //
      rs232_buffer[0]:=read_rs232(250); // get response 13
      if read_error then exit           // no response
      rs232_buffer[1]:=read_rs232(10);  // response message length
      for i:=2 to rs232_buffer[1]+1 do
        begin
         rs232_buffer[i]:=read_rs232(10); // get data
         if read_error then exit          // no response
        end;
    end
    else .......                         // surely a NACK check length!!!
```

After succesfully reception following data are in RS232_Buffer

Rs232_buffer	Value	Remarque
[0]	13	Command 13
[1]	7	String length
[2]	8x	Kwp message header byte 1
[3]	F1	Kwp message header byte 2
[4]	ECU Addr.	Kwp message header byte 3
[5]	54	Response to 14 clear DTC
[6]	FF	All group
[7]	00	
[8]	CKS	Checksumm of KWP2000 message

Commande	21
Fonction	Connect to ABS / ECU
Remarque	This command allows a immediat connection to ECU with KWP2000 fast init. This command uses default ECU address An Auto keepalive function is actived sending the tester present message to ECU. The StartComm String is automatically generated by the chip.
Request	21
Pos. Response	21,Length,KWP2000 string
Neg. Response	NACK not connected please turn ignition key on

Exemple command 21 how connecting to ABS via ABS-Diagnosis line

```
Open_COMM;
clearbuffers;
write_rs232(21);                    // send connect command
rs232_buffer[0]:=read_rs232(100);   // get the response with 100 ms timeout
if read_error then exit             // no response

length:=read_rs232(10);              // response message length
for i:=1 to rs232_length do
        begin
         rs232_buffer[i]:=read_rs232(10); // get data
         if read_error then exit          // no response
        end;
```

Commande	22
Fonction	Disconnect from ABS
Remarque	This command allows a immediate disconnection from ECU . This command stops only the keep aliving . The user needs to wait 5 sec. till he begins with a new initializing. After this command the chips is blocks the RS232 communication for 5000 mS to assure a succefully disconnection and to avoid garbage..
Request	22
Pos. Response	ACK (indicates that the keepaliving is stopped)
Neg. Response	NACK ECU was not connected

Exemple command 22 how disconnect from ABS ECU via ABS diagnosis line

```
clearbuffers;
write_rs232(22);                    // send disconnect command
rs232_buffer[0]:=read_rs232(100);   // get the response with 100 ms timeout
if read_error then exit             // no response
if rs232_buffer[0] = ACK
    then sleep(5000)                // wait for disconnecting
    else                            // was not connected !!!
```

Commande	23
Fonction	Communicate with ABS/ECU
Remarque	
Request	Send 23 to mobydic Wait for 06 (ACK) after receiving ACK user has only 2 sec. to continue Send length , data see exemple
Pos. Response	23 , length , data length=0 indicating no data
Neg. Response	NACK (length doesnt match)

This command is used to communicate with connected ABS-ECU in KWP2000 format.

A Normal KWP2000 Message Format

Header Bytes			Data Bytes							checksum
Length	Target	Source	1	2	3	4	5	6	7	CKS / CRC
80+length	ECU address	F1								CKS

Requested format for mobydic2620

Length	Data Bytes						
	1	2	3	4	5	6	7

Note that Checksum is automatically calculated by modydic2620

Mobydic responds with

Header Bytes			Data Bytes							checksum
Length	Target	Source	1	2	3	4	5	6	7	CKS / CRC
80+length	F1	ECU addr.								CKS

Exemple 1 get AY-Sensor Value using RLI

Length	Data Bytes						
	1	2	3	4	5	6	7
2	21	4B					

Mobydic responds after 23 , length with :

Header Bytes			Data Bytes							checksum
Length	Target	Source	1	2	3	4	5	6	7	CKS / CRC
8x	F1	ECU Addr.	62	4B	xx	xx				CKS

Xx,xx = AY-Sensor data

Delphi exemple for getting AY-Sensor using the RLI

```
clearbuffers;
```

```
write_rs232(23);                          // send communicate command
rs232_buffer[0]:=read_rs232(100);         // get the response with 100 ms timeout
if read_error then exit                   // no response
if rs232_buffer[0] = ACK
    then begin
      write_rs232(02);                     // 2 bytes follow
      write_rs232($21);                    // get data by RLI
      write_rs232($4b);                    // RLI for AY
      rs232_buffer[0]:=read_rs232(250); // response 23
      if read_error then exit              // no response
      rs232_buffer[1]:=read_rs232(10);  // response message length
      for i:=2 to rs232_buffer[1]+1 do
        begin
         rs232_buffer[i]:=read_rs232(10); // get data
         if read_error then exit           // no response
        end;
    end
    else                                   // surely a NACK check length!!!
```

After succesfully reception following data are in RS232_Buffer

Rs232_buffer	Value	Remarque
[0]	23	Command 23
[1]	8	String length
[2]	8x	Kwp message header byte 1
[3]	F1	Kwp message header byte 2
[4]	ECU Addr.	Kwp message header byte 3
[5]	61	Response to 21 read by RLI
[6]	4B	RLI for AY
[7]	xx	AY_sensor data
[8]	xx	AY_sensor data
[9]	CKS	Checksumm of KWP2000 message

Exemple 2 Read DTCs by status from ABS/ECU

Send format for mobydic2620

Length	Data Bytes						
	1	2	3	4	5	6	7
4	18	00	FF	00			

Mobydic responds after 23 , length with :

Header Bytes			Data Bytes							checksum
Length	Target	Source	1	2	3	4	5	6	7	CKS / CRC
8x	F1	ECU Addr.	58	01	dd	dd	ss			CKS

```
clearbuffers;
write_rs232(23);                          // send communicate command
rs232_buffer[0]:=read_rs232(100);         // get the response with 100 ms timeout
if read_error then exit                   // no response
if rs232_buffer[0] = ACK
    then begin
```

```
      write_rs232 (04);                  // 4 bytes follow
      write_rs232($18);                  // get DTCs
      write_rs232($00);                  //
      write_rs232($ff);                  // for all group
      write_rs232($00);                  //
      rs232_buffer[0]:=read_rs232(250); // get response 23
      if read_error then exit            // no response
      rs232_buffer[1]:=read_rs232(10);  // response message length
      for i:=2 to rs232_buffer[1]+1 do
        begin
         rs232_buffer[i]:=read_rs232(10); // get data
         if read_error then exit           // no response
        end;
    end
    else .......                          // surely a NACK check length!!!
```

After succesfully reception following data are in RS232_Buffer

Rs232_buffer	Value	Remarque
[0]	23	Command 23
[1]	9	String length
[2]	8x	Kwp message header byte 1
[3]	F1	Kwp message header byte 2
[4]	ECU Addr.	Kwp message header byte 3
[5]	58	Response to 18 read DTC
[6]	1	Number of DTC
[7]	DTChb	DTC High byte
[8]	DTClb	DTC low byte
[9]	Statusof DTC	Status of DTC
[10]	CKS	Checksumm of KWP2000 message

Exemple 3 Clear DTCs in ABS

Send format for mobydic2620

Length	Data Bytes						
	1	2	3	4	5	6	7
3	14	FF	00				

Mobydic responds after 23 , length with :

Header Bytes			Data Bytes							checksum
Length	Target	Source	1	2	3	4	5	6	7	CKS / CRC
8x	F1	ECU Addr.	54	FF	00					CKS

```
clearbuffers;
write_rs232(23);                       // send communicate command
rs232_buffer[0]:=read_rs232(100);      // get the response with 100 ms timeout
if read_error then exit                // no response
if rs232_buffer[0] = ACK
    then begin
      write_rs232 (03);                 // 3 bytes follow
      write_rs232($14);                 // get DTCs
      write_rs232($ff);                 // for all group
```

```
    write_rs232($00);                    //
    rs232_buffer[0]:=read_rs232(250); // get response 23
    if read_error then exit              // no response
    rs232_buffer[1]:=read_rs232(10);  // response message length
    for i:=2 to rs232_buffer[1]+1 do
      begin
       rs232_buffer[i]:=read_rs232(10); // get data
       if read_error then exit           // no response
      end;
  end
  else .......                          // surely a NACK check length!!!
```

After succesfully reception following data are in RS232_Buffer

Rs232_buffer	Value	Remarque
[0]	23	Command 23
[1]	7	String length
[2]	8x	Kwp message header byte 1
[3]	F1	Kwp message header byte 2
[4]	ECU Addr.	Kwp message header byte 3
[5]	54	Response to 14 clear DTC
[6]	FF	All group
[7]	00	
[8]	CKS	Checksumm of KWP2000 message

Commande	30
Fonction	Set ECU Address on FIAT Airbag pin
Remarque	Default setting is $0 Provided for version 2.0
Request	
Pos. Response	ACK (06)
Neg. Response	N/A

Exemple command 30 how setting the header in mObydic to $25 via Diagnosis line

```
Open_COMM;
clearbuffers;
write_rs232(30);                    // send set address command
write_rs232($25);                   // send address to mOByDic
rs232_buffer[0]:=read_rs232(100);   // get the response with 100 ms timeout
if read_error then exit             // no response
if rs232_buffer[0] = ACK
    then ................           // handle
```

Commande	31
Fonction	Connect to Airbag ECU
Remarque	This command allows a immediat connection to ECU with

	KW82 slow init. This command uses default ECU address or address set by command 30. An Auto keepalive function is actived sending the ACK block message to ECU. Provided for version 2.0
Request	31
Pos. Response	ACK , 6 bytes ISO codes
Neg. Response	NACK not connected please turn ignition key on

Exemple command 31 how connecting to Airbag via Airbag-Diagnosis line

```
Open_COMM;
clearbuffers;
write_rs232(31);                        // send connect command
rs232_buffer[0]:=read_rs232(100);       // get the response with 100 ms timeout
if read_error then exit                 // no response
if rs232_buffer[0] = ACK
    then begin                           // handle
      rs232_buffer[1]:=read_rs232(10);  // get ISOCODE
      rs232_buffer[2]:=read_rs232(10);  //
      rs232_buffer[3]:=read_rs232(10);  //
      rs232_buffer[4]:=read_rs232(10);  //
      rs232_buffer[5]:=read_rs232(10);  //
      rs232_buffer[6]:=read_rs232(10);  //
    end
    else                                // ignition key off !!!
```

Commande	32
Fonction	Disconnect from AIRBAG
Remarque	This command allows a immediate disconnection from ECU . This command Sends a stop block to ECU Provided for version 2.0
Request	32
Pos. Response	ACK
Neg. Response	NACK ECU was not connected

Exemple command 32 how disconnect from Airbag ECU via Airbag diagnosis line

```
clearbuffers;
write_rs232(32);                        // send disconnect command
rs232_buffer[0]:=read_rs232(100);       // get the response with 100 ms timeout
if read_error then exit                 // no response
if rs232_buffer[0] = ACK
    then                                // disconnected
    else                                // was not connected !!!
```

Commande	33
Fonction	Communicate with Airbag ECU
Remarque	Default setting is $0 Provided for version 2.0
Request	Send 33 to mobydic Wait for 06 (ACK) after receiving ACK user has only 2 sec. to continue Send length , data see exemple
Pos. Response	33 , length , data length=0 indicating no data
Neg. Response	NACK (length doesnt match)

This command is used to communicate with connected Airbag ECU in KW82 format.

A Normal KW82 Message Format

Header Bytes		Data Bytes	checksum
Length	Frame Title		
Max 31	command	Max 29	CKS

Requested format for mobydic2620 is the same

Header Bytes		Data Bytes	checksum
Length	Frame Title		
Max 31	command	Max 29	CKS

Note that Checksum is **not** automatically calculated by modydic2620

Mobydic responds with (after 0x33)

Header Bytes		Data Bytes	checksum
Length	Frame Title		
Max 31	command	Max 29	CKS

The handling of message is simplified using a transparent mode . The kw82 message is by passed

Exemple 1 Reading ID code

Header Bytes		Data Bytes	checksum
Length	Frame Title		
2	00	-------	02

Delphi exemple

```
clearbuffers;
write_rs232(33);                        // send communicate command
rs232_buffer[0]:=read_rs232(100);       // get the response with 100 ms timeout
if read_error then exit                 // no response
if rs232_buffer[0] = ACK
    then begin
      write_rs232(02);                   // 2 bytes follow
      write_rs232($00);                  // frame title
      write_rs232($02);                  // checksumm
      rs232_buffer[0]:=read_rs232(250); // response 0x33
      if read_error then exit            // no response
      rs232_buffer[1]:=read_rs232(10);   // response message length
```

```
      for i:=2 to rs232_buffer[1]+1 do
        begin
         rs232_buffer[i]:=read_rs232(10); // get data
         if read_error then exit          // no response
        end;
    end
    else                                 // surely a NACK check length!!!
```

After succesfully reception following data are in RS232_Buffer

Rs232_buffer	Value	Remarque
[0]	33	Response to Command 33
[1]	14	String length
[2]	F6	Response to 00 get ID title
[3]..[14]		ID bytes
[15]	chk	checksum

Exemple 2 Reading Event memory

Header Bytes		Data Bytes	checksum
Length	Frame Title		
2	07	-------	09

Delphi exemple

```
clearbuffers;
write_rs232(33);                      // send communicate command
rs232_buffer[0]:=read_rs232(100);     // get the response with 100 ms timeout
if read_error then exit               // no response
if rs232_buffer[0] = ACK
    then begin
      write_rs232(02);                // 2 bytes follow
      write_rs232($07);               // frame title
      write_rs232($09);               // checksumm
      rs232_buffer[0]:=read_rs232(250); // response 0x33
      if read_error then exit         // no response
      rs232_buffer[1]:=read_rs232(10);  // response message length
      for i:=2 to rs232_buffer[1]+1 do
        begin
         rs232_buffer[i]:=read_rs232(10); // get data
```

```
        if read_error then exit          // no response
      end;
  end
  else                                 // surely a NACK check length!!!
```

After succesfully reception following data are in RS232_Buffer

No event registered !!

Rs232_buffer	Value	Remarque
[0]	33	Response to Command 33
[1]	2	String length
[2]	FC	Response to 07
[3]	chk	checksum

Exemple 3 Reading Snapshoot (crash record 1)

Header Bytes		Data Bytes	checksum
Length	Frame Title		
2	12	01	15

Delphi exemple

```
clearbuffers;
write_rs232(33);                        // send communicate command
rs232_buffer[0]:=read_rs232(100);       // get the response with 100 ms timeout
if read_error then exit                 // no response
if rs232_buffer[0] = ACK
    then begin
      write_rs232(03);                   // 3 bytes follow
      write_rs232($12);                  // frame title
      write_rs232($01);                  // snapshoot
      write_rs232($15);                  // checksumm
      rs232_buffer[0]:=read_rs232(250); // response 0x33
      if read_error then exit            // no response
      rs232_buffer[1]:=read_rs232(10);  // response message length
      for i:=2 to rs232_buffer[1]+1 do
        begin
         rs232_buffer[i]:=read_rs232(10); // get data
         if read_error then exit          // no response
```

```
      end;
   end
   else                                  // surely a NACK check length!!!
```

After succesfully reception following data are in RS232_Buffer

Rs232_buffer	Value	Remarque
[0]	33	Response to Command 33
[1]	N-1	String length
[2]	F4	Response to 12
[3…..N-1]		Parameter 1….n
[N]	chk	checksum

Exemple 4 Clearing

Header Bytes		Data Bytes	checksum
Length	Frame Title		
02	05		07

Delphi exemple

```
clearbuffers;
write_rs232(33);                        // send communicate command
rs232_buffer[0]:=read_rs232(100);       // get the response with 100 ms timeout
if read_error then exit                 // no response
if rs232_buffer[0] = ACK
    then begin
      write_rs232(02);                   // 2 bytes follow
      write_rs232($05);                  // frame title
      write_rs232($07);                  // checksumm
      rs232_buffer[0]:=read_rs232(250); // response 0x33
      if read_error then exit            // no response
      rs232_buffer[1]:=read_rs232(10);  // response message length
      for i:=2 to rs232_buffer[1]+1 do
        begin
         rs232_buffer[i]:=read_rs232(10); // get data
         if read_error then exit          // no response
        end;
   end
   else                                  // surely a NACK check length!!!
```

After succesfully reception following data are in RS232_Buffer

Rs232_buffer	Value	Remarque
[0]	33	Response to Command 33
[1]	02	String length
[2]	09	ACK title from ECU
[3]	0B	checksum

Commande	40
Fonction	Communicate via CAN BUS
Remarque	Reserved for future FIAT cars with CAN BUS Provided for version 2.0
Request	44 , 10 CAN data bytes
Pos. Response	44 , 10 CAN Data bytes
Neg. Response	N/A

Exemple command 44 get RPM via RLI (we guess it will be handled so in future ☺)

```
Open_COMM;
clearbuffers;
write_rs232(40);                    // send CAN  command
write_rs232($07);                   //
write_rs232($E0);                   // ID for PCM
write_rs232($02);                   // PCI
write_rs232($21);                   // get RLI
write_rs232($30);                   // RLI = 30
write_rs232($00);                   // must be filled with 00
write_rs232($00);                   //
write_rs232($00);                   //
write_rs232($00);                   //
write_rs232($00);                   //

repeat

  rs232_buffer[0]:=read_rs232(1000);     // get the response 44
  if read_error then exit                // no response   ....
      for i:=1 to 10 do
        begin
         rs232_buffer[i]:=read_rs232(10); // get data fix length
         if read_error then exit          // no response
```

```
        end;
until ......
```

Rs232_buffer	Value	Remarque
[0]	44	Response to Command 44
[1]..[2]	07 E8	Source ID
[3]..[10]	04 61 30 dd dd 00 00 00	Incoming CAN data

Dd dd = RPM