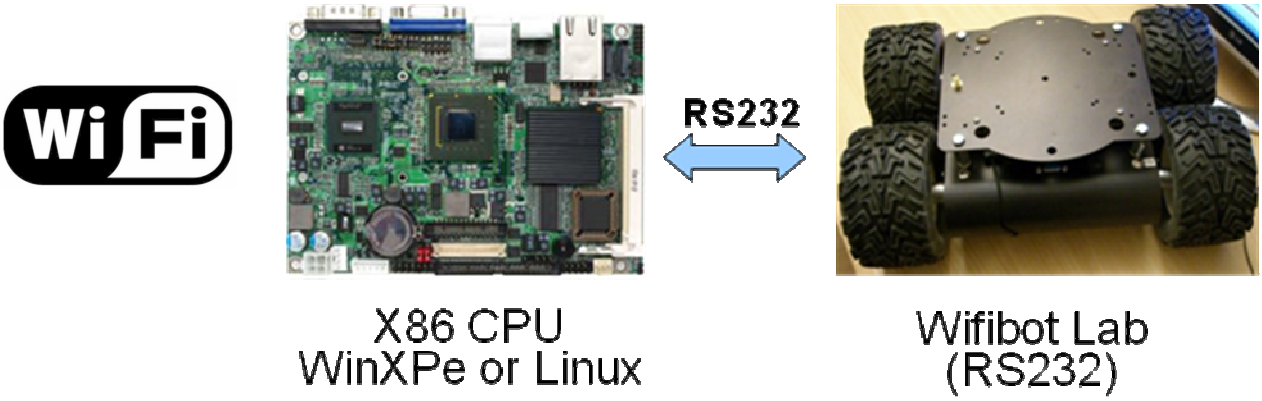
RAW PROTOCOLE using RS232 Protocol encapsulation

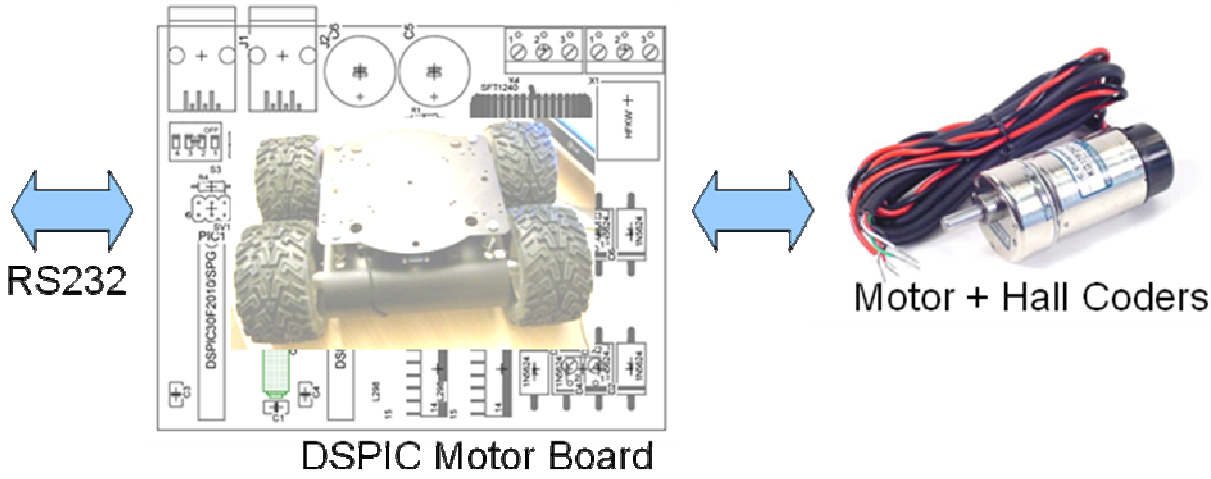
To control the chassis using Ethernet over Wifi you need to send via TCP or UDP at port 15020 (TCP) or 15000/15010 (UDP) special char that will be received by the robot server and forward to the DSPIC 33F using RS232.

To get the data from the chassis it is the same you need to receive TCP or UDP from the robot server some char to read the IR sensors value, the battery level, the robot consumed current, the speed (in tics) and the accumulated Odometry (in tics).

Sending or receiving data is only a matter of sending and receiving chars on Ethernet using sockets.

The Raw protocol is in fact exactly like the RS232 protocol. We have only encapsulate exactly the same data in sockets frame.





TCP :

The computer-WIFIBOT control interface protocol:

Here the communication takes place through one TCP socket. The robot is the server and the interface the client. The communication channel is through robot's **port 15020** and unlike the other protocol it is used for both sending and receiving data.

Messages to port **15020**:

You need to send 9 char using a TCP socket:

Char 1 is 255
Char2 is size (here is 0x07)
Char 3-4 is the left speed 0 -> 240 tics max
Char 5-6 is the right speed 0 -> 240 tics max
Char 7 is the Left / Right speed command flag : Forward / Backward and speed control left & right ON / OFF.
Char 7 is decomposed as follow (1 byte char -> 8 bits):

(128) Bit 7 Left Side Closed Loop Speed control :: 1 -> ON / 0 -> OFF
(64) Bit 6 Left Side Forward / Backward speed flag :: 1 -> Forward / 0 -> Reverse
(32) Bit 5 Right Side Closed Loop Speed control :: 1 -> ON / 0 -> OFF
(16) Bit 4 Right Side Forward / Backward speed flag :: 1 -> Forward / 0 -> Reverse
(8) Bit 3 PID speed 0 is 50 ms 1 is 10 ms (50 ms is recommended)
(4) Bit 2 Not Used (Relay 3 On/Off (future option))
(2) Bit 1 Not Used (Relay 2 On/Off (future option))
(1) Bit 0 Not Used Relay 1 for CPU or sensors. On/Off: 0 is OFF 1 is ON (DSUB9 POWER Pin 3)

Char 8-9 is the CRC 16 bits (char 7 low char 8 high, see end of document for details) not used in TCP

So to control for example the left side and the right side (Left & Right 2 motors, 2 encoders):

The speed is between 0-240

If we want the left side & right side to move at speed 120 forward without motor control we send:
Char 1 is 255
Char2 is 0x07
Char 3 is 0
Char 4 is 120
Char 5 is 0
Char 6 is 120
Char 7 is 80 (0 + 64 + 0 + 16)
Char 8 – Char 9 = CRC16(data) not used in TCP

To GET DATA from the chassis:

The robot server sends you 21 chars using TCP sockets on port 15020:

```
//DSPIC C code to show you the sending part to the PC:

DSPIC send first a '255' char to help synchronise your software and after
(the '255' is not sent in TCP or UDP):

bufsend[0]=(-speedlab);
bufsend[1]=(-speedlab >> 8);//speed is a short and it is tics /50ms or 10ms
bufsend[2]=(unsigned char)(tmpadc2 >> 2);//Bat Volt:10.1V 1.28V 404/4->101
bufsend[3]=(unsigned char)(tmpadc4 >> 2);//3.3v->255 2v-> 624/4 -> 156
bufsend[4]=(unsigned char)(tmpadc3 >> 2);//3.3v->255 2v-> 624/4 -> 156
bufsend[5]=bufposition[0];//Acumulated odometrie is a float
bufsend[6]=bufposition[1];//12ppr x 4 x 51 gear box = 2448 tics/wheel turn
bufsend[7]=bufposition[2];
bufsend[8]=bufposition[3];
bufsend[9]=(speedlab2);
bufsend[10]=(speedlab2 >> 8);
bufsend[11]=(unsigned char)(tmpadc0 >> 2);
bufsend[12]=(unsigned char)(tmpadc1 >> 2);
bufsend[13]=bufposition2[0];
bufsend[14]=bufposition2[1];
bufsend[15]=bufposition2[2];
bufsend[16]=bufposition2[3];
bufsend[17]=0;//robot current // *0.194-37.5 = I in Amp / * 10 for the GUI
: 10 -> 1 A (ACS712 30Amp chip)
bufsend[18]=14; //firmware version
bufsend[19]=crc16 low;
bufsend[20]=crc16 high;
```

You receive on your pc those 21 chars like this:

```
int rcvnbr = recv(socket_handler,sbuf,21,0);

dataL->SpeedFront=(int)((sbuf[1] << 8) + sbuf[0]);
if (dataL->SpeedFront > 32767) dataL->SpeedFront=dataL-
>SpeedFront-65536;
dataL->BatLevel=sbuf[2];
dataL->IR=sbuf[3];
dataL->IR2=sbuf[4];
dataL->odometry=((((long)sbuf[8] << 24))+(((long)sbuf[7] <<
16))+(((long)sbuf[6] << 8))+((long)sbuf[5]));

dataR->SpeedFront=(int)(sbuf[10] << 8) + sbuf[9];
if (dataR->SpeedFront > 32767) dataR->SpeedFront=dataR-
>SpeedFront-65536;
dataR->BatLevel=0;
dataR->IR=sbuf[11];
dataR->IR2=sbuf[12];
dataR->odometry=((((long)sbuf[16] << 24))+(((long)sbuf[15] <<
16))+(((long)sbuf[14] << 8))+((long)sbuf[13]));
dataL->Current=sbuf[17];
dataR->Current=sbuf[17];
dataL->Version=sbuf[18];
dataR->Version=sbuf[18];
```

UDP:

Here the communication takes place through two channels of **UDP sockets**. The first channel (robot's **port 15000**) is for sending commands to the robot while second (robot's **port 15010**) is to request data from it. Communication is done by the use of different and specific messages that the robot separately processes in order to take the proper action. We will present here only the basic set of messages necessary for your program to deal with the control interface, please refer to the source code for more detailed information.

Messages to port **15000 to control the robot**:

You need to send 9 char using a UDP socket:

Char 1 is 255
Char2 is size (here is 0x07)
Char 3-4 is the left speed 0 -> 240 tics max
Char 5-6 is the right speed 0 -> 240 tics max
Char 7 is the Left / Right speed command flag : Forward / Backward and speed control left & right ON / OFF.
Char 7 is decomposed as follow (1 byte char -> 8 bits):

(128) Bit 7 Left Side Closed Loop Speed control :: 1 -> ON / 0 -> OFF
(64) Bit 6 Left Side Forward / Backward speed flag :: 1 -> Forward / 0 -> Reverse
(32) Bit 5 Right Side Closed Loop Speed control :: 1 -> ON / 0 -> OFF
(16) Bit 4 Right Side Forward / Backward speed flag :: 1 -> Forward / 0 -> Reverse
(8) Bit 3 PID speed 0 is 50 ms 1 is 10 ms (50 ms is recommended)
(4) Bit 2 Not Used (Relay 3 On/Off (future option))
(2) Bit 1 Not Used (Relay 2 On/Off (future option))
(1) Bit 0 Not Used Relay 1 for CPU or sensors. On/Off: 0 is OFF 1 is ON (DSUB9 POWER Pin 3)

Char 8-9 is the CRC 16 bits (char 7 low char 8 high, see end of document for details)

So to control for example the left side and the right side (Left & Right 2 motors, 2 encoders):

The speed is between 0-240

If we want the left side & right side to move at speed 120 forward without motor control we send:
Char 1 is 255
Char2 is 0x07
Char 3 is 0
Char 4 is 120
Char 5 is 0
Char 6 is 120
Char 7 is 80 (0 + 64 + 0 + 16)
Char 8 – Char 9 = CRC16(data)

Messages to port **15010 to get data from the robot**:

Here you need to send a sequence of char:

Send 4 char : "**|i|n|i|t|**" and this has to be answered once with "**|o|k|**" to let the control interface know we are really there

Continuously send : "**|d|a|t|a|**" This needs to be answered with21 char of robot data:

```
//DSPIC C code to show you the sending part to the PC:

DSPIC send first a '255' char to help synchronise your software and after
(the '255' is not sent in TCP or UDP):

bufsend[0]=(-speedlab);
bufsend[1]=(-speedlab >> 8); //speed is a short and it is tics / 50 ms
bufsend[2]=(unsigned char)(tmpadc2 >> 2);//Bat Volt:10.1V 1.28V 404/4->101
bufsend[3]=(unsigned char)(tmpadc4 >> 2);//3.3v->255 2v-> 624/4 -> 156
bufsend[4]=(unsigned char)(tmpadc3 >> 2);//3.3v->255 2v-> 624/4 -> 156
bufsend[5]=bufposition[0];//Acumulated odometrie is a float
bufsend[6]=bufposition[1];//12ppr x 4 x 51 gear box = 2448 tics/wheel turn
bufsend[7]=bufposition[2];
bufsend[8]=bufposition[3];
bufsend[9]=(speedlab2);
bufsend[10]=(speedlab2 >> 8);
bufsend[11]=(unsigned char)(tmpadc0 >> 2);
bufsend[12]=(unsigned char)(tmpadc1 >> 2);
bufsend[13]=bufposition2[0];
bufsend[14]=bufposition2[1];
bufsend[15]=bufposition2[2];
bufsend[16]=bufposition2[3];
bufsend[17]=0;//robot current // *0.194-37.5 = I in Amp / * 10 for the GUI
: 10 -> 1 A (ACS712 30Amp chip) // *0.129-25 =I (for ACS712 20A chip)
bufsend[18]=14; //firmware version
bufsend[19]=crc16 low;
bufsend[20]=crc16 high;
```

You receive on your pc those 21 chars like this:

```
int rcvnbr = recvfrom(udp_socket_handler,sbuf,21,0);

dataL->SpeedFront=(int)((sbuf[1] << 8) + sbuf[0]);
if (dataL->SpeedFront > 32767) dataL->SpeedFront=dataL-
>SpeedFront-65536;
dataL->BatLevel=sbuf[2];
dataL->IR=sbuf[3];
dataL->IR2=sbuf[4];
dataL->odometry=((((long)sbuf[8] << 24))+(((long)sbuf[7] <<
16))+(((long)sbuf[6] << 8))+((long)sbuf[5]));

dataR->SpeedFront=(int)(sbuf[10] << 8) + sbuf[9];
if (dataR->SpeedFront > 32767) dataR->SpeedFront=dataR-
>SpeedFront-65536;
dataR->BatLevel=0;
dataR->IR=sbuf[11];
dataR->IR2=sbuf[12];
dataR->odometry=((((long)sbuf[16] << 24))+(((long)sbuf[15] <<
16))+(((long)sbuf[14] << 8))+((long)sbuf[13]));
dataL->Current=sbuf[17];
```

```
            dataR->Current=sbuf[17];
            dataL->Version=sbuf[18];
            dataR->Version=sbuf[18];

In sbuf[20] and sbuf[19] you have the 16bits CRC
Witch is important in UDP.
```

CRC 16 bits (you should not use the starting bit “255”):

```
short Crc16(unsigned char *Adresse_tab , unsigned char Taille_max)
{
      unsigned int Crc = 0xFFFF;
      unsigned int Polynome = 0xA001;
      unsigned int CptOctet = 0;
      unsigned int CptBit = 0;
      unsigned int Parity= 0;

      Crc = 0xFFFF;
      Polynome = 0xA001;
      for ( CptOctet= 0 ; CptOctet < Taille_max ; CptOctet++)
      {
            Crc ^= *( Adresse_tab + CptOctet);

            for ( CptBit = 0; CptBit <= 7 ; CptBit++)
            {
                  Parity= Crc;
                  Crc >>= 1;
                  if (Parity%2 == true) Crc ^= Polynome;
            }
      }
      return(Crc);
}
```