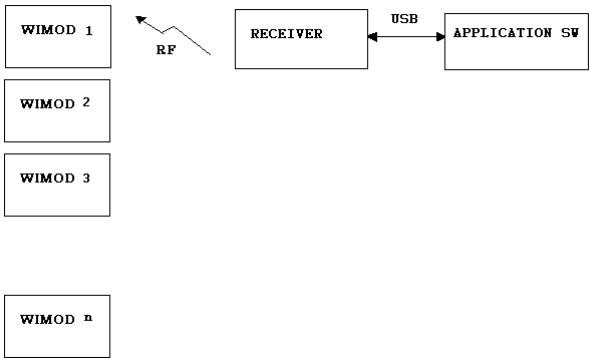
WIMOD Communication protocol

The communication protocol between a WIMOD modules network and a receiver sw is on 2 levels.

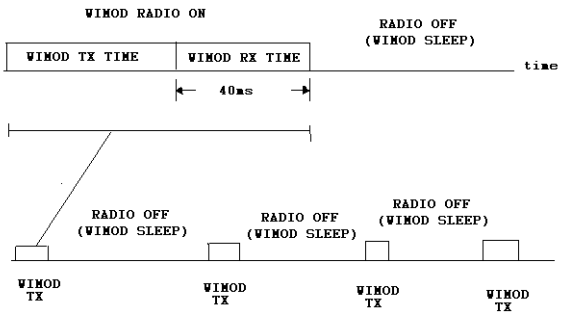


First level is the communication between the application sw and the receiver RF module.
You need to implement this level to initialize the receiver hardware and to get RF data coming from the WIMODs network.
It is based on a USB port managed by a FTDI FT232R chip. The port is seen as a virtual serial port so no special driver or library are necessary to use in your driver. You have just to use the standard API for a serial port.
You must open the serial port with the followings parameters : baud: 19200 : parity : none : stop bit : 1 : N. bit : 8
The second level of the communication protocol is necessary to implement commands to WIMODs and to decode data coming from WIMODS.

WIMOD /RECEIVER RF COMMUNICATION

Each WIMODs network is identified by network address (4 chars) (common to all modules) and for each RF module is assigned a dedicated address (4 chars) (normally the last four digits of its serial number).
These allow that can exist more than one WIMODs networks at the same time
These addresses (network and module address) are AEP factory assigned.
AEP will communicate for each delivered system the addresses that will be assigned to each modules

From the RF communication point of view all WIMODs modules are slaves and the RF receiver is a master.
Each WIMODs modules transmit its load values at regular interval (from .1s to 5s) . The time between two data can be changed by the master.
The master from time to time (let me say : every 5 s) must send at least one command to each slave. In this way WIMOD recognize that the receiver is on and continue to transmit.
To save battery life each WIMOD actives the radio only for a certain time. It transmit the load data packet and then for 40ms still keep the radio on waiting for a master command.
The master so must send its commands inside the RX TIME slot. As soon as it receives some data from a WIMOD it must send its command to it to be sure the WIMOD radio is still on.



If a WIMOD does not receive at least one command from the master it goes in power down mode in order to save battery life.
In power down mode WIMOD transmit a data packet every 8s.

RECEIVER INITIALITATION

To init the RF receiver module it is necessary to send some commands to it.

```
"C151";                 // send * after command
"C01XXXX";              // XXXX = network address
"C02YYYY";              // YYYY = master address
"C0406";                // the data packed length is 6 byte
"C07Z";                 // Z= power Level (valid values are : '0'-'1'-'2'-'3'
"C08";                  // Init Radio
"C14";                  // Set Output *WD
"C150";                 // don't send * after command
```

It is not the scope of this document describe this commands. Refere to the document **G3P_um_rel_201b.pdf**

MASTER COMMANDS

There are just a few commands that the master can send to each WIMOD to set parameters :
The application sw must send 3 messages to the RF receiver module with the indication of the destination WIMOD address command

```
"C03KKKK";              // KKKK = destination WIMOD address slave address
"C30XXXX00";            // XXXX00 : 6 byte of Payload data command
"C31";
```

Inside the KKKK field you have to specify the destination WIMOD and inside the XXXX00 you put the commands and its associate parameters (last 2 char always 0).

The meaning of XXXX commands fields is the following :

```
X X X X
+----------------- command parameter     P1
   +-------------- command parameter     P2
      +---------- command parameter     P3
         +------- command specifier :  valid values are   '0': do nothing : used for keep alive the WIMOD
                                                          '1': zero on/off:          P1 = '0' → zero Off  : '1' → zero On        : ASCII value
                                                          '2': set power level :     P1 = '0' → power level = -10dbm             : ASCII Value
                                                                                     P1=  '1' → power level = -2dbm
                                                                                     P1=  '2' → power level = +6dbm
                                                                                     P1 = '3' → power level = +10dbm
                                                          '3': TxRate Interval (in step of 100ms)
                                                                                     P1= 1..50                                   : binary value
                                                                                      P2=0                                       ; binary value
                                                                                      P3=0                                       ; binary value
                                                          '6': Set Filter            P1=0..31                                    : binary value
```

DECODING RECEIVED DATA FROM WIMOD

The application sw must polls data coming from WIMODs modules on the serial line
Any time the RF receiver module get valid data from a WIMOD it send a data packet of 10 characters.

The first 4 characters are the address of the WIMOD module.
The following 6 characters are the load data sent from the WIMOD.

The 6 chars received from the WIMOD are bit oriented with the following format :

byte 5 byte 4 byte 3 byte 2 byte 1 byte 0

7 0 7 0 7 0 7 0 7 0 7 0 bit
xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
|||||||| |||||||| |||||||| ||||++++++++++++++++++++++++ 20 bit binary value with sign
|||||||| |||||||| |||||||| |+++------------------------ Moltiplicative Factor
|||||||| |||||||| |||||||| +-------------------------- Zero On / Zero Off
|||||||| |||||||| |||||||+---------------------------- Low Battery
|||||||| |||||||| |||||++----------------------------- Level Power
|||||||| |||||||| ||||+------------------------------- AEP reserved
|||||||| |||||||| ++++-------------------------------- AEP reserved
|||||||| ++++++++-- Filter Value
++++++++--- Tx Rate

Moltiplicative Factor:
3 bytes: gives an exponent to the 20bit value according to the following table
000 = 0.0001
001 = 0.0010
010 = 0.0100
011 = 0.1000
100 = 1.00
101 = 10.0
110 = 100.0
111 =1000.0

So the real load value is
Displayed Value ActualValue*FattoreMoltiplicativo

CODE EXAMPLE to decode receiving data from a WIMOD

RxBuffer is the RX communication buffer.
j is the index inside the buffer where it was recognized the message coming from a WIMOD address 'E0E2' .
In this example you have

RxBuffer[j+0]= 'E'
RxBuffer[j+1]= '0'
RxBuffer[j+2]= 'E'
RxBuffer[j+3]= '2'

```
char RxBuffer[10000];
int stato;              //signal if the load cell is overload/underload/  in range    state
float value;            // WIMOD Load
int PowerLevel;         // WIMOD actual setup : Power Level,Filter,TxRate
int Filtro;
int TxRate;

void DecodeMessageWIMOD(int j)
{
union
{
  long L;
  BYTE Buffer[4];
} L;

        L.Buffer[0]=RxBuffer[4+j];                      // extract the load info (20 bit)
        L.Buffer[1]=RxBuffer[5+j];
        L.Buffer[2]=RxBuffer[6+j] & 0xf;
        if (RxBuffer[6+j] & 0x8)                        // detect the sign and extend it to the long variable
        {
                L.Buffer[3]=0xff;
                L.Buffer[2]|=0xf0;
        }
        else
        {
                L.Buffer[3]=0;
                L.Buffer[2]&=0x0f;
        }

        if (L.L==0x7ffff) stato=1;                      //UPPER    (if  value = 0x7ffff then WIMOD is in overload)
        else if (L.L==0xfff80000) stato=2;              //LOWER   (if value =   0xfff80000 then is underload)
        else
        {
                stato=0;
                switch(RxBuffer[6+j] & 0x70)
                {
                case 0x00:FactMul=0.0001f;break;
                case 0x10:FactMul=0.001f;break;
                case 0x20:FactMul=0.01f;break;
                case 0x30:FactMul=0.1f;break;
                case 0x40:FactMul=1.0f;break;
                case 0x50:FactMul=10.0f;break;
                case 0x60:FactMul=100.0f;break;
                case 0x70:FactMul=1000.0f;break;
                }
                value=(float)L.L*FactMul;
        }
```

```
if (RxBuffer[6+j] & 0x80)
        ZeroOn=true;
else
        ZeroOn=false;

if (RxBuffer[7+j] & 0x1)
        LowBattery=true;
else
        LowBattery=false;

Filtro=RxBuffer[8+j];
TxRate=RxBuffer[9+j];
switch(RxBuffer[7+j] & 0x06)
{
case 0x00:PowerLevel=0;break;
case 0x02:PowerLevel=1;break;
case 0x04:PowerLevel=2;break;
case 0x06:PowerLevel=3;break;
}
```