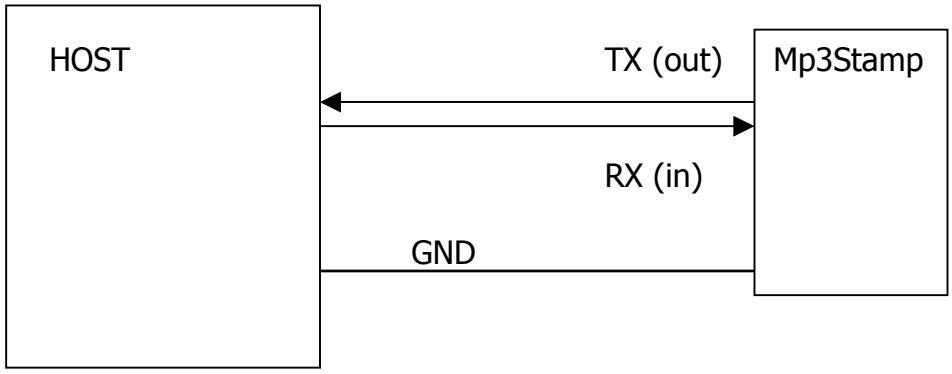
# Software Guide

Mp3Stamp modules have been developped by Reality sys to allow programmers easily add voice, sound effects or HiFi quality music to a lot of applications, from leisure to security, through mobiles applications and signage.
One single module integrating all read and decoding functionnalities makes easier hardware and softwarer design.
Mp3Stamp realizes reading operations on standard memory cards pre-recordered on PC/Windows or MAC, and assumes decoding up to the analogic output ready to be amplified or directly available for a headphone output.

For hardware ressources, « host » application only needs a serial port to manage the Mp3Stamp.
As it is a low speed communication, it is also possible to have a “software” serial connection if an UART is not available.
Mp3Stamp discharges the main processor of all real time processes linked to audio decoding.
Initialization is stand alone, just send 2 bytes to start reading and decoding of a file on the memory card.
Moreover, Mp3Stamp has its own adjustment switches and a memory to save configuration, that reduce again the main processor tasks.

## 1 - HARDWARE INTERFACE REALIZATION

Communication is assumed through an asynchronous serial connection configured at 9600 bauds, 8 data bits, no parity, 1 or 2 stop bits.
If the module is directly connected to a micro-controler, it is enough to connect directly RX & TX input and output of Mp3Stamp to the micro-controler.
The module runs at 3.3V but his input allow 5V level, it could be then connected to a 5V micro-controler.
To connect the module to a RS232 or RS485 equipment, it is necesary to envisage an adapter with adequate levels.



## 2 – SOFTWARE INTERFACE REALIZATION

MP3Stamp control is done via commands transmit from host to the module
There are 2 commands format :

FORMAT 1 : <COMMAND BYTE>
FORMAT 2 : <COMMAND BYTE><PARAM BYTE>

In first case, command is constitued of a unique byte, in the second case, command byte is followed with a paramter byte.
There is no particular byte at the end of the command (<CR><LF>).

In its more simple release, the communication can be one-way, host to module, in this case only the RX(in) signal is needed.
Most commands do not produce answer from the module.

It could be necessary to send request to the module and use TX(out) signal.
Some commands are envisaged to give module status. These commands are at format 1, answers are usually made of 4 or 8 successive bytes transmitted directly after the command.

Interface parameters

| | | |
|---|---|---|
| MODE BYTE | 1BYTE | R/W |
| NAME | 8BYTES | R |
| TIME | 4BYTES | R |
| CURNT TRACK | 1BYTE | R/W |

MODE BYTE: status bits and configuration
NAME : contains current file name (MSDOS mode 8 characters)
TIME : on stop contains current track duration, on playing remaining track duration.
CURNT TRACK : current track number, it is possible to modify this pointer during playing without affecting the on course play.

Initialization

The initialization is automatic and do not require any host intervention.
On powering ON, MP3Stamp firmware detects the memory card presence, go through the root directory, and initialize status bits from presets saved in flash memory.
LED stay lighted during this test.
This operation is 1 to 2 sec last, that depends of presence or not of the memory card, and the number of files on the card.
If the memory card is not detected, the module emits a beep on the audio output and the LED stay lighted. This error is seen on the status byte.

If the memory card is detected, a character string describing file system is directly transmitted, and if the card contains at least 1 compatible file the LED light OFF.

Sample :

```
Partition type :6
Partition start:32
MSWIN4.1
Sect/cluster  :4
Root :279
```

String end with the &FF character (255).

Starting there, the board is ready to receive commands.
If a memory card is detected, current track pointer is positionned to track 1 or to a predefined track specified in a BOOT command.

If CONTINUE bit of MODEBYTE specify it, track playing start.

On power ON or after a RESET command, host processor has to wait up to the end of automatic initialization before sending commands, for that, it could wait a few seconds or wait to receive the 255 character which follows the partition description.

The first command should be (but it is not requested) a status request to assume that a memory card is detected and check the number of files.
Empty the host recpetion's buffer before send this first command.

Play a track

A few commands allow to play a track.
Most simple is the PLAY command ( « O » character as open), this one do not need parameter, current track is played.
« E » command execute the track as the one supplied as parameter.
Otherwise, « D » command modify paramter value to CRNT TRACK without start playing.
Rem : first track is track 1, there is no track 0.

Synchronize playing with Host

Host processor can ask periodically the MODEBYTE (bit 7) to know if playing is ended. It could also ask the remaining duration before the end (command « T »)
Lastly, a byte is automatically transmitted at the end of a track playing, this byte represents MODEBYTE.

Les codes des commandes sont donnés en ASCII et en base 10.

<table>
<tr><td colspan="3">Commands with 1 parameter, 1 BYTE of command, 1 BYTE parameter,<br>No answer</td></tr>
<tr><td>GOTO N :<br>« D »,BYTE<br>68,1-255</td><td colspan="2">Get the current track on N (1-255).<br>If track N does not exist, go back to track 1.<br>Do not start Playing but after this command,<br>SNDTIME may be used and return back the total duration.</td></tr>
<tr><td>PLAY N :<br>« E »,BYTE<br>69,1-255</td><td colspan="2">Get the current track on N (1-255)<br>Start Playing.<br>If track N does not exist, go back to track 1<br>And do not start Playing.</td></tr>
<tr><td>SETVOL N :<br>« V », BYTE<br>86,0-255</td><td colspan="2">Set value N (0-255) to the audio level output<br>Value is DIRECTLY saved into flash<br>0 is max. volume, 255 is the minimum.<br>at 80 output is pratically not hearable</td></tr>
<tr><td rowspan="10">SETMODE B :<br>« M »,BYTE<br>77,0-63</td><td colspan="2">Modify function mode, each bit of B has a signification</td></tr>
<tr><td>B0 : Continue</td><td>0: Playing stop at the end of a track, nothing start at powering on.<br>1: Playing continue at the end of a track, start again at the first after the last<br>Track X is played at powering on.</td></tr>
<tr><td>B1 : Loop</td><td>0: tracks continue automatically.<br>1: only 1 track is repeated.</td></tr>
<tr><td>B2 : Vol LOCAL</td><td>1: volume level may not be modified by the board<br>0: volume level is controlled by the board</td></tr>
<tr><td>B3 : Trig LOCAL</td><td>1: retrigger during playing is impossible<br>0: normal mode</td></tr>
<tr><td>B4 : Led<br>! write only</td><td>0: Led OFF<br>1: Led ON (use this function when power off)</td></tr>
<tr><td>B5: Sort</td><td>0: Normal 1:Indexed (A0000)</td></tr>
<tr><td>B6 ! read only</td><td>1: ERR MMC CARD</td></tr>
<tr><td>B7 ! read only</td><td>0 : STOP<br>1 : PLAYING</td></tr>
<tr><td colspan="2">Byte board value is DIRECTLY saved into flash</td></tr>
</table>

| Commands requesting an answer, 1 BYTE of command,<br>4 or 8 BYTES of answer | |
|---|---|
| SNDSTAT : « S »<br>83 | Status request, answer return back Player status<br>4 BYTES returned back<br><br><mode><volume><current track><files nbr><br><br><mode> seer SETMODE<br><volume> 0=max to 255=mute<br><piste> 1 to files nbr (<=255)<br><nbfiles> number of files on the memory card |
| SNDNAME :« I »<br>73 | Request titles of current track<br>8 BYTES returned back (ASCII) |
| SNDTIME : « T »<br>84 | Return back time code within 1/100 of seconds.<br>- on stop following D command, total duration.<br>- on PLAY or PAUSE, reminding duration.<br>4 BYTES returned back, lighter weight byte first<br>It is possible to read only the first 2 bytes (if < 10 min) |
| LIST : « L »<br>76 | List all files on the card and their size |

Rem : to make LOOP effective, CONTINUE mode is to be activated.

***ANNEXE :* Sample of source code using the protocole.**

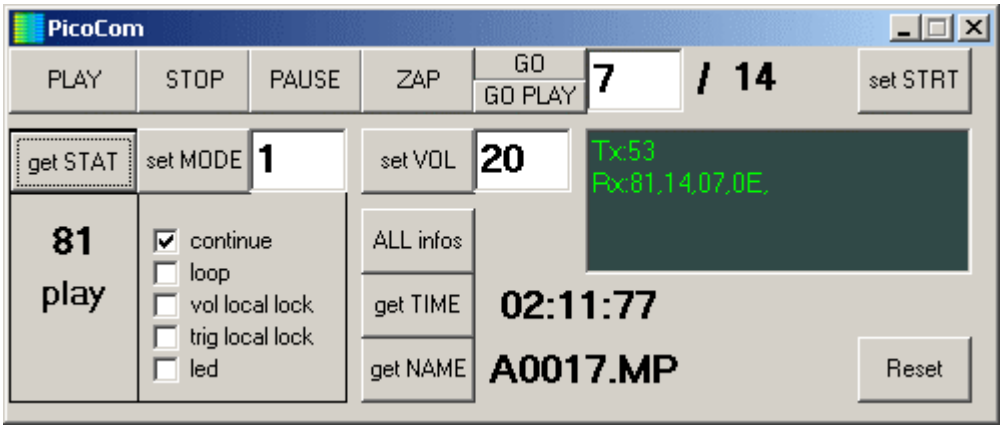
```
//---------------------------------------------------------------
//---------------------------------------------------------------
// PLAY
void __fastcall TForm1::Button1Click(TObject *Sender)
{
        Memo1->Text="Tx:" + IntToHex(byte('O'),2);
        ComPort1->Send("O");
}
//---------------------------------------------------------------
// PAUSE
void __fastcall TForm1::Button2Click(TObject *Sender)
{
        Memo1->Text="Tx:" + IntToHex(byte('!'),2);
        ComPort1->Send("!");
}
//---------------------------------------------------------------
// STOP
void __fastcall TForm1::Button4Click(TObject *Sender)
{
        Memo1->Text="Tx:" + IntToHex(byte('H'),2);
        ComPort1->Send("H");
}
//---------------------------------------------------------------
// ZAP
void __fastcall TForm1::Button10Click(TObject *Sender)
{
        Memo1->Text="Tx:" + IntToHex(byte('Z'),2);
        ComPort1->Send("Z");
}
//---------------------------------------------------------------
// GOTO N
void __fastcall TForm1::Button5Click(TObject *Sender)
{    char n;
             n=Edit1->Text.ToInt();
             Memo1->Text="Tx:" + IntToHex(byte('D'),2)+ "," +IntToHex(n,2);
             ComPort1->Send("D");
             ComPort1->Send(n);
}
//---------------------------------------------------------------
// GOTO N et PLAY
void __fastcall TForm1::Button12Click(TObject *Sender)
{
   char n;
             n=Edit1->Text.ToInt();
             Memo1->Text="Tx:" + IntToHex(byte('E'),2)+ "," +IntToHex(n,2);
             ComPort1->Send("E");
             ComPort1->Send(n);
}
//---------------------------------------------------------------
// SET VOLUME N
void __fastcall TForm1::Button3Click(TObject *Sender)
{    char n;
             n=Edit2->Text.ToInt();
```

```
            Memo1->Text="Tx:" + intToHex(byte('V'),2)+","+IntToHex(n,2);
            ComPort1->Send("V");
            ComPort1->Send(n);
}
//---------------------------------------------------------
// SET MODE N
void __fastcall TForm1::Button9Click(TObject *Sender)
{    char n;
            n=Edit3->Text.ToInt();
            ComPort1->Send("M");
            ComPort1->Send(n);
}
//---------------------------------------------------------
// SET STARTING TRACK
void __fastcall TForm1::Button13Click(TObject *Sender)
{
            Memo1->Text="Tx:" + IntToHex(byte('B'),2);
            ComPort1->Send("B");

}
//---------------------------------------------------------
// SOFT RESET
void __fastcall TForm1::Button14Click(TObject *Sender)
{
        Memo1->Text="Tx:" + IntToHex(byte('X'),2);
        ComPort1->Send("X");

}
//---------------------------------------------------------
// REQUETE STATUS
void __fastcall TForm1::Button6Click(TObject *Sender)
{        last_cmnd=1;
         Memo1->Text="Tx:" + IntToHex(byte('S'),2);
         ComPort1->Send("S");
}
//---------------------------------------------------------
// REQUETE TITRE
void __fastcall TForm1::Button7Click(TObject *Sender)
{        last_cmnd=3;
         Memo1->Text="Tx:" + IntToHex(byte('I'),2);
         ComPort1->Send("I");
}
//---------------------------------------------------------
// REQUETE TEMPS
void __fastcall TForm1::Button8Click(TObject *Sender)
{       last_cmnd=2;
        Memo1->Text="Tx:" + IntToHex(byte('T'),2);
        ComPort1->Send("T");
}
```

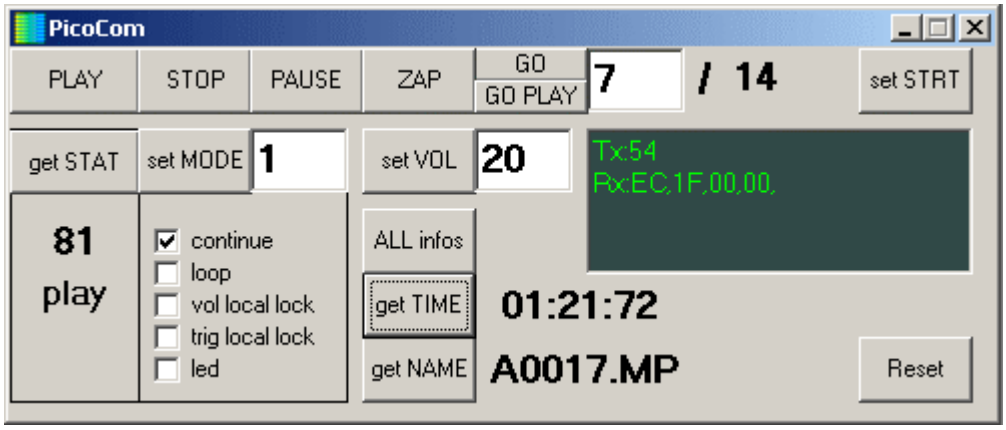
```
//---------------------------------------------------------------
//---------------------------------------------------------------
// Interruption réception série
void __fastcall TForm1::ComPort1ReceiveCallBack(AnsiString Data)
{       int temp;
        int sec;
        int ms;
        int min;
        AnsiString s;
        char* cp=new char[16]; // buffer de réception

        if(last_cmnd!=0){

        cp=Data.c_str();
        Memo1->Lines->Add("Rx:");

        for(temp=0;temp<4;temp++) Memo1->Text=
            Memo1->Text+IntToHex(byte(cp[temp]),2)+",";
// STATUS

        if(last_cmnd==1) {
                Label1->Caption =  IntToHex(byte(cp[0]),2);     // MODE B
                Edit2->Text =  byte(cp[1]);                     // VOLUME
                Edit1->Text =  byte(cp[2]);                     // TRACK nb
                Label2->Caption =  byte(cp[3]);                 // NBFILES

// Mise à jour des bits de mode
                CheckBox1->Checked=((cp[0] & 1)==1);
                CheckBox2->Checked=((cp[0] & 2)==2);
                CheckBox3->Checked=((cp[0] & 4)==4);
                CheckBox4->Checked=((cp[0] & 8)==8);
                CheckBox5->Checked=((cp[0] & 16)==16);

                if((cp[0] & 128)==128)
                                {Label6->Caption="play";}
                        else {Label6->Caption="stop";};
                if((cp[0] & 64)==64) {Label6->Caption="ERR";};

                };

// TIME, seul les 2 premiers bytes utilisés

      if(last_cmnd==2){             temp=byte(cp[0])+256*byte(cp[1]);
                sec=temp/100;
                ms=temp-sec*100;
                min=sec / 60;
                sec=sec % 60;
                s.sprintf ("%2.2u:%2.2u:%2.2u",min,sec,ms);
                Label4->Caption=s;
                };

// TITRE
         if(last_cmnd==3){                  for(temp=4;temp<8;temp++)
         Memo1->Text=Memo1-Text+IntToHex(byte(cp[temp]),2)+",";
                Label5->Caption =Data;};
        };

        last_cmnd=0;
        wait_reponse=false;
        delete[] cp;
}
```

```
//---------------------------------------------------------------
//---------------------------------------------------------------
// Changement des check_box des bits de MODE
void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{if (CheckBox1->Checked ) {mode_byte=mode_byte | 1;}
 else {mode_byte=mode_byte & 254;}  ;
Edit3->Text=int(mode_byte);
}
//---------------------------------------------------------------
void __fastcall TForm1::CheckBox2Click(TObject *Sender)
{if (CheckBox2->Checked ) {mode_byte=mode_byte | 2;}
 else {mode_byte=mode_byte & 253;}  ;
Edit3->Text=int(mode_byte);
}
//---------------------------------------------------------------
void __fastcall TForm1::CheckBox3Click(TObject *Sender)
{if (CheckBox3->Checked ) {mode_byte=mode_byte | 4;}
 else {mode_byte=mode_byte & 251;}  ;
Edit3->Text=int(mode_byte);
}
//---------------------------------------------------------------
void __fastcall TForm1::CheckBox4Click(TObject *Sender)
{if (CheckBox4->Checked ) {mode_byte=mode_byte | 8;}
 else {mode_byte=mode_byte & 247;}  ;
Edit3->Text=int(mode_byte);
}
//---------------------------------------------------------------
void __fastcall TForm1::CheckBox5Click(TObject *Sender)
{if (CheckBox5->Checked ) {mode_byte=mode_byte | 16;}
 else {mode_byte=mode_byte & 239;}  ;
Edit3->Text=int(mode_byte);
}
//---------------------------------------------------------------
//---------------------------------------------------------------
// Demande toutes les infos; status,nom,durée
// utilise un timer pour enchainer les 3 commandes
void __fastcall TForm1::Button11Click(TObject *Sender)
{
  Button11->Enabled=false;
  com_step=1;
  wait_reponse=true;
  Button6Click(Form1); // STAT
  Timer1->Interval =10;
  Timer1->Enabled =true;
}
// timer pour la commande all infos
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{  if (wait_reponse==false){
          wait_reponse=true;
          if (com_step==1)  Button8Click(Form1); // TIME
          if (com_step==2)  Button7Click(Form1);// NAME
          com_step++;
        };

   if (com_step==3) {Timer1->Enabled =false;
   Button11->Enabled=true;};

}
//--FIN-----------------------------------------------------------
```

Send of a command SNDSTAT ('S' or 83 or 53h)
Answer : 4 bytes MODE = 81h
VOL = 14h (20)
TRACK = 07h (7)
nbTRACKS = 0Eh (14)




Send of a command SNDTIME ('T' or 84 or 54h)
Answer : 4 bytes 00 00 1F EC = 8172 * 0.01s




Send of a command PLAY 9 ('E' or 69 or 45h)