Gestion du port série (COM) sous Windows

Pour piloter des applications à partir d'un ordinateur le port série présente de nombreux avantages en particulier il est très bien protégé contre les fausses manœuvres et autorise la connexion et déconnexion lorsque l'ordinateur est sous tension (pour les machines récentes ,ce n'est pas vrai pour la plupart les vieux PC équipés de microprocesseurs antérieurs au Pentium, fonctionnant sous DOS) , enfin un courant d'intensité notable (10mA) est disponible sur chaque broche et peut servir à alimenter de petites applications.

Le port série utilise un connecteur 9 broches (DB9), souvent 25 broches pour COM2 , dont les accès sont les suivants

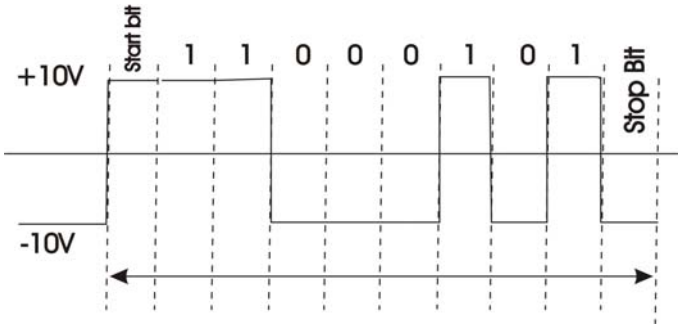
Broche DB9	Broche DB25	Sens	Nom	Fonction
1		Entrée	DCD(Data Carrier Detect)	Détection porteuse
2	3	Entrée	RXD ou RD(Receive Data)	Réception
3	2	Sortie	TXD ou TD(Transmet Data)	Emission
4	20	Sortie	DTR(Data Terminal Ready)	Terminal prêt
5	7		Masse	
6	6	Entrée	DSR(Data set ready)	Emission prête
7	4	Sortie	RTS(Request to Send)	Demande d'émission
8	5	Entrée	CTS(Clear to send)	Prêt à emettre
9		Entrée	RI(Ring indicator)	Sonnerie

La norme RS232 qui est associé à ce type de port est très complexe et rarement utilisée en électronique. Le plus souvent l'échange d'octets s'effectue sur les seuls fils RXD et TXD . Dans ce cas les signaux d'échange doivent être simulés , DSR relié à DTR et RTS relié à CTS . Le câble de liaison entre deux ordinateurs ou entre ordinateur et application est alors le suivant :

Il faut se souvenir que les signaux disponibles sur ce port n'ont pas le niveau TTL .
Etat de repos 1 typiquement -10V (entre -3 et -15V=
Etat de travail 0 typiquement +10V (entre 3 et 15V)

L'envoi d'un octet commence par un bit de start (niveau travail) suivi des 8 bits de l'octet suivis ou non d'un bit de parité et terminé par 1 1;1/2 ou 2 bits de stop (niveau repos) .

Par exemple pour l'envoi de C5H = 11000101



Les vitesses de transmission sont normalisées 110 150 300 600 1200 2400 4800 9600 19200 bits/secondes (Bauds)

Pour l'association avec un microprocesseur une conversion de ce signal aux niveaux TTL est nécessaire .Les circuits MAX232 ou équivalents sont parfaitement adaptés à ce travail. Alimentés en 5V ils fabriquent en interne le ±10V nécessaire . (circuit à pompe de charge)

.

Sous DOS

Pour développer des applications électroniques les vieilles machines incapables d'accueillir les logiciels récents sont encore bien pratiques

Dans ce type de micro ordinateur le circuit de gestion de la liaison série est un UART (8250) contenant un certain nombre de registres.

A l'émission ce sont les registres dans lequel se trouve le caractère à émettre (Tampon d'émission) et celui qui contient l'octet en cours d'émission .

A la réception on trouve de même un registre dans lequel se trouve l'octet en cours de réception et celui contenant le dernier caractère reçu. .(Tampon de réception).

Le transfert est défini par:

- Un registre de définition de la vitesse et des paramètres nécessaires
- Un registre d'état.

Le registre d'Initialisation se trouve à l'adresse $03BF (pour COM1 , $02BF pour le COM2) il a la configuration représentée dans les tableaux suivants:

Le bit 7 joue un rôle important, il permet l'accès au registre définissant la vitesse .Ce registre de vitesse de 16 bits se trouve localisé aux adresses $3F8 et $3F9 mais n'est accessible que si le bit précédent est à 1 .

En fonction de la vitesse désirée le mot à charger est indiqué dans le tableau ci contre.:

Vitesse = F_0 / Baud rate divisor	
Vitesse (Bauds)	Baud rate divisor
1200	$0060
2400	$0030
4800	$0018
9600	$000C
19200	$0006

Le registre d'adresse $3FB définit le format du transfert son contenu est détaillé dans le tableau suivant :

Contenu du registre 3FB

7	6	5	4	3	2	1	0	3FB
						0 0 1 1	0 1 0 1	**Nombre de bits de données** 5 6 7 8
					0 1	⇒ ⇒	⇒ ⇒	**Bits de Stop** 1 bit de stop Si 5 bits de données 1,5 bit de stop Autre cas 2 stops bits
				0 1	⇒ ⇒	⇒ ⇒	⇒ ⇒	**Parité** pas de bit de parité 1 bit de parité
			0 1	⇒ ⇒	⇒ ⇒	⇒ ⇒	⇒ ⇒	**Type de parité** Parité impaire Parité paire
		0 1	⇒ ⇒	⇒ ⇒	⇒ ⇒	⇒ ⇒	⇒ ⇒	**Parité fixe** parité dépendant de l'octet (Si bit 3 à 1) Parité fixe
	0 1	⇒ ⇒	⇒ ⇒	⇒ ⇒	⇒ ⇒	⇒ ⇒	⇒ ⇒	**Break forcé** inutilisé envoie 0 sur tous les bits à émettre
0 1	⇒ ⇒	⇒ ⇒	⇒ ⇒	⇒ ⇒	⇒ ⇒	⇒ ⇒	⇒ ⇒	**Accès au registre de vitesse** accès aux autres registres accès au registre de vitesse

Le registre d'état se trouve à l'adresse $03FD

0	1 si un octet est dans le tampon de réception
1	1 si erreur d'écrasement
2	1 si erreur de parité (**Attention ce bit est RAZ par la lecture**)
3	1 si erreur de trame
4	1 si détection d'un break

5	1 si le tampon d'émission est vide
6	1 si le registre d'émission est inoccupé

Si le flag de réception (bit 0) est à 1 cela veut dire qu'une donnée peut être lue dans le tampon de réception . Il faut la récupérer puis remettre ce flag à 0 avant que la donnée suivante vienne écraser la précédente.

Quand le flag d'émission (Bit 5) est à 1 l'envoi de la donnée est terminée Après avoir mis ce flag à 0 on peut placer la donnée suivante dans le tampon d'émission.

Le registre tampon qui sert à l'émission et la réception est à l'adresse $03F8, il s'agit d'un registre qui se trouve à la même adresse que le registre de vitesse mais s'en distingue par l'état du bit 7 de $3FD .L'accès à ce tampon exige que ($3FD)-7 (Le DLBA) soit au 0.

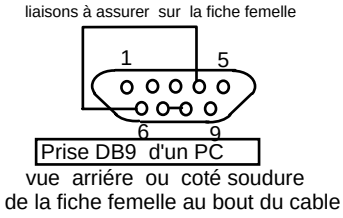
La gestion du transfert peut se faire par interruption ou plus aisément par scrutation du registre d'état.

Le registre d'adresse 3F9 accessible avec DLAB=0 contient des informations sur les interruptions associées au circuit 8250 .Dans certains cas comme nous le verrons plus loin il est souhaitable de désactiver ces interruptions en chargeant 00000000 dans ce registre.

Attention :Si on utilise seulement 3 fils RXD TXD et Masse, il relier faut ensembles

- Request to Send et Clear to Send (RTS-CTS)
- Data Set Ready et Data Terminal Ready (DSR-DTR)

SIGNAL	DB25	DB9
Masse	7	5
Emission TXD	2	3
Réception RXD	3	2
RTS	4	7
CTS	5	8
DSR	6	6
DTR	20	4



D'ou le logiciel de gestion: (en Pascal)

Initialisation :

```
Port[$3FB]:=$80;          accès au registre de vitesse
PortW[$3F8]:=$000C;       Init 9600 bauds
Port[$3FB]:=$03;          Init 8b,1bs,NoP
Residu:=Port[$3F8];       Pour vider le tampon . Residu est un octet
```

Lecture d'un octet arrivant vers le PC

```
Repeat
Res:=Port[$3FD] AND $01;      Attente d'arrivée
Until Res=$01;                bit 1 de 3FD
DATA:=Port[$08]               La donnée lues'appelle DATA
```

Envoi d'un caractère sur le bus

```
Repeat
Res:=( Port[$3FD] AND $20 );      Attente de tampon
Until Res=$20                     d'émission vide bit 5 de 3FD;
Port[$3F8]:=DATA       ;          DATA est l'octet à envoyer
```

Sous Windows

A partir de Windows 95 , pour des raisons de sécurité , l'accès aux ports est impossible directement, il faut utiliser les routines du système d'exploitation ce qui rend l'utilisation du port série bien plus délicate. C'est pour cette raison que nombre d'applications qui tournaient sous DOS ne fonctionnent plus sur les machines modernes.

Si la manipulation des routines de Windows écrites en C n'est pas à la portée de tout le monde , certains développeurs ont fait heureusement le travail pour nous et il existe des DLL (Dynamic Links Librairy) exploitable par des programmes tels que DELPHI , VISUAL BASIC ou VISUAL C++ qui permettent d'accéder au port série.

Pour DELPHI la DLL la plus connue est **TComPort** , elle est parfaitement adaptée à la gestion du port série suivant la norme RS232, un peu moins bien pour la gestion bit par bit qui intéresse l'électronicien. Un programmeur allemand Burkhard Kainka a écrit en 1999 un livre traitant de ces questions qui est malheureusement épuisé , mais les DLL nécessaires pour la gestion du port série contenues dans un fichier **Rscom.DLL** sont téléchargeable sur son site .Ces routines sont particulièrement bien adaptées à un usage électronique, elles permettent non seulement l'échange d'octets par un processus type RS232 mais aussi la gestion individuelle des bornes du port.

Un exemple simple est donné par Yoann dans http://delphipage.free.fr/portserie.html , c'est d'ailleurs grâce à cet article que j'ai découvert rscom .

La DLL est constituée de plusieurs fonctions et procédures ,parmi lesquelles celles qui sont les plus importantes pour gérer les bornes du port sont les suivantes :

Function OPENCOM	ouverture du port série et définition des paramètres
Procedure CLOSECOM	fermeture de la liaison
Procedure SENDBYTE(dat;integer);	Envoi d'un octet
Procedure READBYTE ; integer	lecture d'un octet reçu
Procedure CLEARBUFFER	effacement du contenu du buffer interne
Procedure DTR	commande le la broche DTR
Procedure RTS	commande le la broche RTS
Procedure TXD	commande le la broche TXD
Function CTS	lecture de l'état de la broche CTS
Function DSR	lecture de l'état de la broche CTS
Function DCD	lecture de l'état de la broche DCD
Function RI	lecture de l'état de la broche RI
Procedure DELAY(real)	C'est l'équivalent de la procédure DELAY du Pascal 5 sous DOS

Les paramètres en entrée ou sortie des procédures et fonctions sont parfois un peu surprenantes, ainsi les procédures DTR RTS TXD reçoivent un mot (WORD) alors que le fil ne peut se trouver que dans 2 états; état 0 (niveau +15V) pour une entrée 0 et état 1 (-15V) pour une entrée quelconque différente de 0 .

Les fonctions ou procédures nécessaires doivent être déclarées au début du chapitre implémentation avec les mots réservés **stdcall;external 'Rscom.DLL',** ***elles doivent impérativement se trouver le répertoire ou sont crées les fichiers de Delphi.***

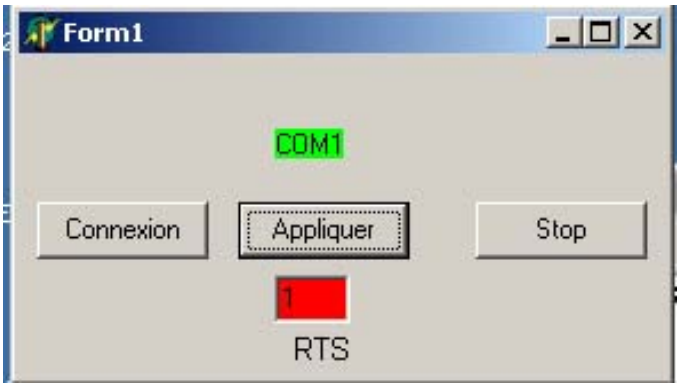
Liens:	
Pour RSCOM	http://www.b-kainka.de/download.htm
Pour TComPort :	http://sourceforge.net/projets/comport/
Renseignements sur Delphi:	http://delphipage.net
	http://delphi.developpez.com
	http://delphipage.free.fr/portserie.html
	http://www.delphifr.com
Le cours de J-C Armici	http://www.unvrai.com/cours.php

Function OPENCOM (OpenString:Pchar);integer;**stdcall;external** 'Rscom.DLL ' ;

```
Function  OPENCOM ( OpenString:Pchar);integer;stdcall;external 'Rscom.DLL ' ;
Procedure  CLOSECOM();stdcall;external 'Rscom.DLL ' ;                    n
Procedure  SENDBYTE( dat;integer); stdcall;external 'Rscom.DLL ' ;
Procedure READBYTE():integer; stdcall;external 'Rscom.DLL ' ;
Procedure CLEARBUFFER();stdcall;external 'Rscom.DLL ' ;
Procedure DTR(d:DWORD); stdcall;external 'Rscom.DLL ' ;
Procedure RTS(d:DWord); stdcall;external 'Rscom.DLL ' ;
Procedure TXD((d;DWord); stdcall;external 'Rscom.DLL ' ;
Function  CTS:integer; stdcall;external 'Rscom.DLL ' ;
Function DSR:integer; stdcall;external 'Rscom.DLL ' ;
Function  DCD:integer; stdcall;external 'Rscom.DLL ' ;
Function  RI:integer; stdcall;external 'Rscom.DLL ' ;
Procedure DELAY( DelayTime:real); stdcall;external 'Rscom.DLL ' ;
```

Exemple de pilotage des bornes du port.

Pour commander l'état d'une broche du port (en sortie) il suffit de déclarer la procédure nécessaire et d'utiliser un bouton et une zone de saisie (Edit) .Par exemple :



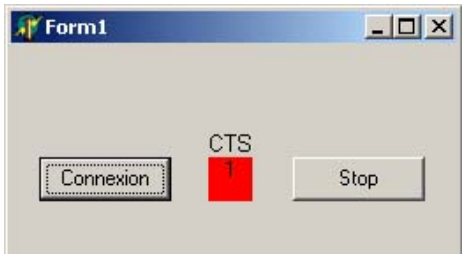
Le projet ci contre est constitué outre des commandes permettant d'activer ou de désactiver le port COM1, d'une zone de saisie où l'on inscrit la valeur que l'on souhaite transmettre à la broche RTS (1 par défaut) , un bouton qui commande l'envoi (appliquer) et une zone colorée matérialisant l'état obtenu (rouge +10V , vert -10V)

Le programme est le suivant :

```
unit out_RTS;

interface

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

type
  TForm1 = class(TForm)
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    Edit1: TEdit;
    Label1: TLabel;
    Label2: TLabel;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
  private
    { Déclarations privées }
  public
    { Déclarations publiques }
  end;

var
  Form1: TForm1;
```

```
implementation

{$R *.DFM}
Function OPENCOM(OpenString:PChar):integer;stdcall;external 'RScom.DLL';
procedure CLOSECOM();stdcall;external 'RSCom.DLL';
Procedure RTS(d:word);stdcall;external 'RSCom.DLL';
      // inclusion des éléments nécessaires tirés de la DLL


procedure TForm1.Button1Click(Sender: TObject);
begin
OPENCOM('COM1:baud=2400 parity=N data:=8 stop=0');
Label2.color:=clLIME;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
CLOSECOM();
Label2.color:=clRED;

end;

procedure TForm1.Button3Click(Sender: TObject);
var a:integer;
begin
a:=strTOint(Edit1.text);
RTS(a);
if a=0 then Edit1.color:=clLIME else Edit1.Color:=clRED;
end;

end.
```

Pour la lecture de l'état des broches d'entrée l'utilisation d'un Timer est le plus commode, à chaque événement Timer l'état de la broche est testée et le résultat transféré dans une zone de lecture. C'est le cas du projet ci-dessous. Le Timer étant défini avec **interval=10** l'examen de l'état a lieu toutes les 10mS .



Extrait du programme :

```
Function OPENCOM(OpenString:PChar):integer;stdcall;external 'RScom.DLL';
procedure CLOSECOM();stdcall;external 'RSCom.DLL';
Function CTS:integer;stdcall;external 'RScom.DLL';
      // inclusion des éléments nécessaires tirés de la DLL

procedure TForm1.Timer1Timer(Sender: TObject);
var a:integer;
begin
a:=CTS;                        //appel fonction CTS
Label1.caption :=intTOstr(a);          //affichage  résultat
if a=0 then label1.color:=clLime
      else Label1.color:=clRed;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
OPENCOM('COM1:baud=2400 parity=N data:=8 stop=0');
Timer1.enabled:=True;
```

```
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
CLOSECOM();

end;
```

Echange d'octets par TXD RXD

On utilise les fonctions et procédures SENDBYTE et READBYTE , pour l'envoi pas de problèmes SENDBYTE(a) envoie l'octet a sur COM1 avec le format défini préalablement par OPENCOM .

A la réception c'est la fonction READBYTE qui est utilisée, mais attention elle ne fait que lire le contenu d'un buffer interne dans lequel sont venus se stocker les octets arrivés sur le port.

Le plus simple est d'utiliser un Timer qui périodiquement vient lire ce buffer en lançant la fonction READBYTE..Si le buffer est vide la valeur lue est -1 , sinon l'octet qui a été reçu.(0 à 255)

Si le port reçoit des octets plus rapidement que le timer ne vient les lire, ils s'entassent dans le buffer où ils peuvent ensuite être lus l'un après l'autre.(si la capacité du buffer n'est pas dépassée). Imaginons par exemple que le port reçoive 25 octets à raison de 10 par seconde alors que le Timer a un intervalle de une seconde. Les 25 octets sont acquis en 2,5 secondes puis sont lus et affichés l'un après l'autre à raison de 1 par seconde pendant 25 secondes.

La procédure CLEARBUFFER permet de vider le buffer .

Comme exemple le projet suivant gère l'échange d'octets entre deux ordinateurs. Un timer interne lit le buffer d'entrée 20 fois par seconde , (un octet à 2400bauds est transmis en 4mS environ) ce qui limite la vitesse de l'échange à 20 octets par seconde , le buffer étant vidé après chaque lecture .

(en accélérant le timer, une cadence de plus de 100 octets par seconde aurait été possible)

Les octets reçus s'affichent à gauche dans la fenêtre jaune , les octets à envoyer doivent être introduits dans la zone de saisie en vert à droite .Le bouton Envoi déclenche la procédure d'envoi. .

Les deux boutons supérieurs ouvrent et ferment la liaison .

Le programme:

```
unit echange;

interface

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

type
  TRecoit = class(TForm)
    Button2: TButton;
    Label5: TLabel;
    Timer1: TTimer;
    Edit2: TEdit;
```

```
    Label1: TLabel;
    Edit3: TEdit;
    Label2: TLabel;
    Button1: TButton;
    Label3: TLabel;
    Edit1: TEdit;
    Button3: TButton;
    Label4: TLabel;
    Label6: TLabel;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Timer1Timer(Sender:TObject);
    procedure Button3Click(Sender: TObject);


  private
  a,b,c,d:integer;
    { Déclarations privées }
  public

    { Déclarations publiques }
  end;

var
  Recoit: TRecoit;

implementation

{$R *.DFM}    // les procédures et fonctions issues de la DLL
 function OPENCOM(OpenString:PChar):integer;stdcall;external 'RSCOM.DLL';
 procedure CLOSECOM();stdcall;external 'RSCOM.DLL';
procedure SENDBYTE(dat:integer);stdcall;external 'RSCOM.DLL';
 function READBYTE():integer;stdcall;external 'RSCOM.DLL';
procedure CLEARBUFFER();stdcall;external 'RSCOM.DLL';

procedure TRecoit.Button1Click(Sender: TObject);     // Connexion
begin
OpenCOM('COM1: baud=2400 parity=N data=8 stop=1');
ClearBuffer();
Edit1.Color:=clLime;
timer1.enabled:=True;           // au départ seul le bouton d'ouverture de la liaison est visible
Edit3.visible:=True;
Edit2.visible:=True;
Button3.visible:=True;
Label1.visible:=True;
Label2.visible:=True;
Label4.visible:=True;
Label5.visible:=True;
Label6.visible:=True;
end;

procedure TRecoit.Button2Click(Sender: TObject);     // déconnexion
begin
CloseCom;
Edit1.color:=clRED;
Timer1.enabled:=False;
Edit3.visible:=False;
Edit2.visible:=False;
Button3.visible:=False;
Label1.visible:=False;
```

```
Label2.visible:=False;
Label4.visible:=False;
Label5.visible:=False;
Label6.visible:=False;
end;

Procedure TRecoit.Timer1Timer(Sender:TObject);   // le Timer lit le buffer et modifie les
Begin                                            //couleurs
a:=ReadByte();
if a=-1 then edit2.color:=clred
else Begin
     edit2.color:=clGreen;
    Label4.caption:=intTOstr(a);
     ClearBuffer();
End;
 End;

procedure TRecoit.Button3Click(Sender: TObject);   // envoi d'un octet
begin
 a:=strTOint(Edit3.Text);
 SENDBYTE(a);
 a:=a+1;
 if a>255 then a:=0;
 Edit3.Text:=intTOstr(a);
end;

end.
```