```pascal
{ SdpoSerial v0.1.4

  CopyRight (C) 2006-2010 Paulo Costa

  This library is Free software; you can rediStribute it and/or modify it
  under the terms of the GNU Library General Public License as published by
  the Free Software Foundation; either version 2 of the License, or (at your
  option) any later version.

  This program is diStributed in the hope that it will be useful, but WITHOUT
  ANY WARRANTY; withOut even the implied warranty of MERCHANTABILITY or
  FITNESS FOR A PARTICULAR PURPOSE. See the GNU Library General Public License
  for more details.

  You should have received a Copy of the GNU Library General Public License
  along with This library; if not, Write to the Free Software Foundation,
  Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

  This license has been modified. See File LICENSE.ADDON for more inFormation.
  Should you find these sources without a LICENSE File, please contact
  me at paco@fe.up.pt
}

unit SdpoSerial;

{$mode objfpc}{$H+}

interface

uses
{$IFDEF LINUX}
  Classes,
{$IFDEF UseCThreads}
  cthreads,
{$ENDIF}
{$ELSE}
  Windows, Classes, //registry,
{$ENDIF}
  SysUtils, synaser,  LResources, Forms, Controls, Graphics, Dialogs;


type
  TBaudRate=(br___300, br___600, br__1200, br__2400, br__4800, br__9600, br_19200,
             br_38400, br_57600, br115200, br230400, br460800, br921600);
  TDataBits=(db8bits,db7bits,db6bits,db5bits);
  TParity=(pNone,pOdd,pEven,pMark,pSpace);
  TFlowControl=(fcNone,fcXonXoff,fcHardware);
  TStopBits=(sbOne,sbTwo);

  TModemSignal = (msRI,msCD,msCTS,msDSR);
  TModemSignals = Set of TModemSignal;

const
  ConstsBaud: array[TBaudRate] of integer=(
    300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200,
    230400, 460800, 921600);

  ConstsBits: array[TDataBits] of integer=(8, 7 , 6, 5);
  ConstsParity: array[TParity] of char=('N', 'O', 'E', 'M', 'S');
  ConstsStopBits: array[TStopBits] of integer=(SB1, SB2);


```

```pascal
type
  TSdpoSerial = class;

  TComPortReadThread=class(TThread)
  public
    MustDie: boolean;
    Owner: TSdpoSerial;
  protected
    procedure CallEvent;
    procedure Execute; override;
  published
    property Terminated;
  end;

  { TSdpoSerial }

  TSdpoSerial = class(TComponent)
  private
    FActive: boolean;
    FSynSer: TBlockSerial;
    FDevice: string;

    FBaudRate: TBaudRate;
    FDataBits: TDataBits;
    FParity: TParity;
    FStopBits: TStopBits;

    FSoftflow, FHardflow: boolean;
    FFlowControl: TFlowControl;

    FOnRxData: TNotifyEvent;
    ReadThread: TComPortReadThread;

    FAltBaudRate: integer;

    procedure DeviceOpen;
    procedure DeviceClose;

    procedure ComException(str: string);
    function BaudRateValue: integer;

  protected
    procedure SetActive(state: boolean);
    procedure SetBaudRate(br: TBaudRate);
    procedure SetAltBaudRate(altbr: integer);
    procedure SetDataBits(db: TDataBits);
    procedure SetParity(pr: TParity);
    procedure SetFlowControl(fc: TFlowControl);
    procedure SetStopBits(sb: TStopBits);

  public
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;

    procedure Open;
    procedure Close;

    // read data from port
    function DataAvailable: boolean;
    function ReadData: string;
//    function ReadBuffer(var buf; size: integer): integer;

```

```
    // write data to port
    function WriteData(data: string): integer;
    function WriteBuffer(var buf; size: integer): integer;

    // read pin states
    function ModemSignals: TModemSignals;
    function GetDSR: boolean;
    function GetCTS: boolean;
    function GetRing: boolean;
    function GetCarrier: boolean;

    // set pin states
//    procedure SetRTSDTR(RtsState, DtrState: boolean);
    procedure SetDTR(OnOff: boolean);
    procedure SetRTS(OnOff: boolean);
    procedure SetBreak(OnOff: boolean);

  published
    property Active: boolean read FActive write SetActive;

    property BaudRate: TBaudRate read FBaudRate write SetBaudRate; // default br115200;
    property AltBaudRate: integer read FAltBaudRate write SetAltBaudRate; // default br115200;
    property DataBits: TDataBits read FDataBits write SetDataBits;
    property Parity: TParity read FParity write SetParity;
    property FlowControl: TFlowControl read FFlowControl write SetFlowControl;
    property StopBits: TStopBits read FStopBits write SetStopBits;

    property SynSer: TBlockSerial read FSynSer write FSynSer;
    property Device: string read FDevice write FDevice;

    property OnRxData: TNotifyEvent read FOnRxData write FOnRxData;
  end;

procedure Register;

implementation

{ TSdpoSerial }

procedure TSdpoSerial.Close;
begin
  Active:=false;
end;

procedure TSdpoSerial.DeviceClose;
begin
  // flush device
  if FSynSer.Handle<>INVALID_HANDLE_VALUE then begin
    FSynSer.Flush;
    FSynSer.Purge;
  end;

  // stop capture thread
  if ReadThread<>nil then begin
    ReadThread.FreeOnTerminate:=false;
    ReadThread.MustDie:= true;
    while not ReadThread.Terminated do begin
      Application.ProcessMessages;
    end;
    ReadThread.Free;
    ReadThread:=nil;
```

```pascal
  end;

  // close device
  if FSynSer.Handle<>INVALID_HANDLE_VALUE then begin
    FSynSer.Flush;
    FSynSer.CloseSocket;
  end;
end;

constructor TSdpoSerial.Create(AOwner: TComponent);
begin
  inherited;
  //FHandle:=-1;
  ReadThread:=nil;
  FSynSer:=TBlockSerial.Create;
  FSynSer.LinuxLock:=false;
  FHardflow:=false;
  FSoftflow:=false;
  FFlowControl:=fcNone;
  {$IFDEF LINUX}
  FDevice:='/dev/ttyS0';
  {$ELSE}
  FDevice:='COM1';
  {$ENDIF}
//  FBaudRate:=br115200;
  FAltBaudRate := 0;
end;

function TSdpoSerial.DataAvailable: boolean;
begin
  if FSynSer.Handle=INVALID_HANDLE_VALUE then begin
    result:=false;
    exit;
  end;
  result:=FSynSer.CanReadEx(0);
end;

destructor TSdpoSerial.Destroy;
begin
  Close;
  FSynSer.Free;
  inherited;
end;

procedure TSdpoSerial.Open;
begin
  Active:=true;
end;

procedure TSdpoSerial.DeviceOpen;
begin
  FSynSer.Connect(FDevice);
  if FSynSer.Handle=INVALID_HANDLE_VALUE then
    raise Exception.Create('Could not open device '+ FSynSer.Device);

  FSynSer.Config(BaudRateValue(),
                 ConstsBits[FDataBits],
                 ConstsParity[FParity],
                 ConstsStopBits[FStopBits],
                 FSoftflow, FHardflow);

  // Launch Thread
  ReadThread := TComPortReadThread.Create(true);
  ReadThread.Owner := Self;
  ReadThread.MustDie := false;
  ReadThread.start;
end;


function TSdpoSerial.ReadData: string;
begin
  result:='';
  if FSynSer.Handle=INVALID_HANDLE_VALUE then
    ComException('can not read from a closed port.');

  result:=FSynSer.RecvPacket(0);
end;

procedure TSdpoSerial.SetActive(state: boolean);
begin
  if state=FActive then exit;

  if state then DeviceOpen
  else DeviceClose;

  FActive:=state;
end;

procedure TSdpoSerial.SetBaudRate(br: TBaudRate);
begin
  if FSynSer.Handle<>INVALID_HANDLE_VALUE then begin
    FSynSer.Config(BaudRateValue(), ConstsBits[FDataBits], ConstsParity[FParity],
                   ConstsStopBits[FStopBits], FSoftflow, FHardflow);
  end;
  FBaudRate:=br;
end;

procedure TSdpoSerial.SetAltBaudRate(altbr: integer);
begin
  FAltBaudRate := altbr;
  if FSynSer.Handle<>INVALID_HANDLE_VALUE then begin
    FSynSer.Config(BaudRateValue(), ConstsBits[FDataBits], ConstsParity[FParity],
                   ConstsStopBits[FStopBits], FSoftflow, FHardflow);
  end;
end;


procedure TSdpoSerial.SetDataBits(db: TDataBits);
begin
  if FSynSer.Handle<>INVALID_HANDLE_VALUE then begin
    FSynSer.Config(BaudRateValue(), ConstsBits[FDataBits], ConstsParity[FParity],
                   ConstsStopBits[FStopBits], FSoftflow, FHardflow);
  end;
  FDataBits:=db;
end;

procedure TSdpoSerial.SetFlowControl(fc: TFlowControl);
begin
  if fc=fcNone then begin
    FSoftflow:=false;
    FHardflow:=false;
  end else if fc=fcXonXoff then begin
    FSoftflow:=true;
    FHardflow:=false;
  end else if fc=fcHardware then begin
    FSoftflow:=false;
    FHardflow:=true;
  end;

  if FSynSer.Handle<>INVALID_HANDLE_VALUE then begin
    FSynSer.Config(BaudRateValue(), ConstsBits[FDataBits], ConstsParity[FParity],
                   ConstsStopBits[FStopBits], FSoftflow, FHardflow);
  end;
  FFlowControl:=fc;
end;

{
procedure TSdpoSerial.SetFlowControl(fc: TFlowControl);
begin
  if FHandle<>-1 then begin
    if fc=fcNone then CurTermIO.c_cflag:=CurTermIO.c_cflag and (not CRTSCTS)
    else CurTermIO.c_cflag:=CurTermIO.c_cflag or CRTSCTS;
    tcsetattr(FHandle,TCSADRAIN,CurTermIO);
  end;
  FFlowControl:=fc;
end;
}
procedure TSdpoSerial.SetParity(pr: TParity);
begin
  if FSynSer.Handle<>INVALID_HANDLE_VALUE then begin
    FSynSer.Config(BaudRateValue(), ConstsBits[FDataBits], ConstsParity[FParity],
                   ConstsStopBits[FStopBits], FSoftflow, FHardflow);
  end;
  FParity:=pr;
end;

procedure TSdpoSerial.SetStopBits(sb: TStopBits);
begin
  if FSynSer.Handle<>INVALID_HANDLE_VALUE then begin
    FSynSer.Config(BaudRateValue(), ConstsBits[FDataBits], ConstsParity[FParity],
                   ConstsStopBits[FStopBits], FSoftflow, FHardflow);
  end;
  FStopBits:=sb;
end;

function TSdpoSerial.WriteBuffer(var buf; size: integer): integer;
begin
//  if FSynSer.Handle=INVALID_HANDLE_VALUE then
 //   ComException('can not write to a closed port.');
  result:= FSynSer.SendBuffer(Pointer(@buf), size);
end;

function TSdpoSerial.WriteData(data: string): integer;
begin
  result:=length(data);
  FSynSer.SendString(data);
end;


function TSdpoSerial.ModemSignals: TModemSignals;
begin
  result:=[];
  if FSynSer.CTS then result := result + [ msCTS ];
  if FSynSer.carrier then result := result + [ msCD ];
  if FSynSer.ring then result := result + [ msRI ];
  if FSynSer.DSR then result := result + [ msDSR ];
```

```pascal
end;

function TSdpoSerial.GetDSR: boolean;
begin
  result := FSynSer.DSR;
end;

function TSdpoSerial.GetCTS: boolean;
begin
  result := FSynSer.CTS;
end;

function TSdpoSerial.GetRing: boolean;
begin
  result := FSynSer.ring;
end;

function TSdpoSerial.GetCarrier: boolean;
begin
  result := FSynSer.carrier;
end;

procedure TSdpoSerial.SetBreak(OnOff: boolean);
begin
//  if FHandle=-1 then
//    ComException('can not set break state on a closed port.');
//  if OnOff=false then ioctl(FHandle,TIOCCBRK,1)
//  else ioctl(FHandle,TIOCSBRK,0);
end;


procedure TSdpoSerial.SetDTR(OnOff: boolean);
begin
  FSynSer.DTR := OnOff;
end;


procedure TSdpoSerial.SetRTS(OnOff: boolean);
begin
  FSynSer.RTS := OnOff;
end;


procedure TSdpoSerial.ComException(str: string);
begin
  raise Exception.Create('ComPort error: '+str);
end;

function TSdpoSerial.BaudRateValue: integer;
begin
  if FAltBaudRate > 0 then begin
    result := FAltBaudRate;
  end else begin
    result := ConstsBaud[FBaudRate];
  end;
end;

{ TComPortReadThread }

procedure TComPortReadThread.CallEvent;
begin
  if Assigned(Owner.FOnRxData) then begin
```

```
    Owner.FOnRxData(Owner);
  end;
end;

procedure TComPortReadThread.Execute;
begin
  try
    while not MustDie do begin
      if Owner.FSynSer.CanReadEx(100) then
        Synchronize(@CallEvent);
    end;
  finally
    Terminate;
  end;

end;


procedure Register;
begin
  RegisterComponents('5dpo', [TSdpoSerial]);
end;

initialization
{$i TSdpoSerial.lrs}

end.
```