```
{*****************************************************}
{                                                     }
{       SoftReal TAPI Components (Delphi 5)           }
{                                                     }
{       Copyright (c) 2002 SoftReal Kft.              }
{       All rights reserved.                          }
{                                                     }
{       Version:      v1.1                            }
{       Compiler:     Delphi 5                        }
{       System:       Windows 98, Me, NT4, 2K, XP     }
{       Programmers:  2002 L.G.Horvath                }
{       E-mail:       softreal@softreal.hu            }
{                                                     }
{*****************************************************}

unit SrTapi;

{ This module uses the Telephony API interface units Tapi.pas, Tapi.inc and
  Tapi3Err, obtained through the Joint Endeavour of Delphi Innovators
  (Project JEDI). You may retrieve the latest version of these files
  at the Project JEDI home page, located at http://delphi-jedi.org

  The object classes in this unit support only 32bits Windows, with TAPI v2.0
  and higher versions. TAPI30 conditional define should be used (see Tapi.inc).

  v1.1 (12/9/2002):
  TTapiLineDevice is now capable of answering incoming calls.
  The new OnCallOffering event handler is defined for notifying the application
  about an incoming call, along with the caller address (if available).
  The new AnswerCall method may be used for answering the call.
  LINECALLSTATE_OFFERING call state is now converted by tapiCallStateToStr
  (this state is passed by OnCallState before the call to OnCallOffering).

  Note: Since many current modems offer support for Caller ID, but virtually
        every modem disables Caller ID as the default, you may want to use
        an additional setup string to enable caller ID.
        Try AT+CLIP=1 command for GSM modems, AT#CLS=8#CID=1 for voice modems
        and AT#CID=1 (or AT#CID=2, AT%CCID=1, AT%CCID=2, AT+VCID=1, AT#CC1,
        AT*ID1) for all other modems to get a caller ID string from the modem
        along with the first "RING" string (the string will look like
        DATE = 1209 TIME = 1122 NMBR = 1234567 or CLIP: "1234567" - anyhow,
        it's the modem driver's business).

  v1.0 (9/23/2002):
  The first release of this module.
}


interface

uses
  Classes, SysUtils, Contnrs, Windows, Tapi;

{$INCLUDE TAPI.INC}

const
  TAPI_LOW_VERSION = $00010000;
{$IFDEF TAPI30}
  TAPI_HIGH_VERSION = $00030000;
{$ELSE}
  {$IFDEF TAPI22}
    TAPI_HIGH_VERSION = $00020002;
  {$ELSE}
    {$IFDEF TAPI21}
      TAPI_HIGH_VERSION = $00020001;
    {$ELSE}
      {$IFDEF TAPI20}
        TAPI_HIGH_VERSION = $00020000;
      {$ELSE}
        'TAPI versions lower than 2.0 are not supported'
      {$ENDIF}
    {$ENDIF}
  {$ENDIF}
{$ENDIF}

type
{ ETapiError is a new exception class for TAPI errors. }
  ETapiError = class(Exception)
  public
    constructor Create(ACode: Cardinal);
    property Code: Cardinal;
  end;

  TTapiLineService = class;
  TTapiLineDevice = class;

  TLineMsgEvent = procedure (hDevice, dwMsg, dwCallbackInstance,
      dwParam1, dwParam2, dwParam3: Cardinal) of object;

{ TTapiLineDevice is the object class of TAPI line devices (phone devices are
  not supported). }
  TCallConnectedEvent = procedure (Sender: TObject; CommHandle: THandle;
      const CommName, ErrorMsg: string) of object;
  TCallErrorEvent = procedure (Sender: TObject; Code: Cardinal) of object;
  TCallerIDEvent = procedure (Sender: TObject;
      const CallerID: string; Flag: Cardinal) of object;
  TCallStateEvent = procedure (Sender: TObject;
      State, Info, Priv: Cardinal) of object;

  TTapiLineDevice = class(TObject)
  public
    constructor Create(AService: TTapiLineService; AnID: Cardinal);
    destructor Destroy; override;
    procedure Close;
      { Closes the line, sets Active property to False.
        You should not call this method, if line is not Active.
        Changing Active to False will call Close. }
    procedure AnswerCall;
      { Answers the incoming call (reported by OnCallOffering).
        If there's no call then LINEERR_CALLUNAVAIL exception will be raised.
        The communication line will be available in the OnCallConnected
        event handler. OnCallFinished event handler will report the end
        of the call. }
    procedure DropCall;
      { Ends the data-modem call (an outgoing call initiated by MakeCall).
        You may also call it when there is an incoming call reported by
        OnCallOffering, but it will not hang up, the next ring will create
        a new call again. If there is no call then LINEERR_CALLUNAVAIL exception
        will be raised. OnCallFinished event handler will report the end of the
        call. }
    procedure GetCallerID(var ID: string; var Flag: Cardinal);
      { Obtaines the LINECALLINFO structure about the current call
        (by lineGetCallInfo) and retrieves the dwCallerIDXxx data.
        If there is no call then LINEERR_CALLUNAVAIL exception will be raised. }
    function IsDataModem: Boolean;
      { Line supports LINEMEDIAMODE_DATAMODEM media mode. }
    procedure MakeCall(const DestAddress: string);
      { Places a data-modem call on the specified line to the specified
        destination address. The address must follow the standard dialable
        number format. (Default country code and call parameters will be used.)
        Multiple calls at the same time are not supported, LINEERR_INUSE
        exception will be raised when trying to make a new call before
        the previous one has ended. If no exception is raised, the OnCallFinished
        event handler will report the end of the session. The communication line
        will be available in the OnCallConnected event handler.
        The application must be prepared for handling errors, since we
        can't prevent the line from receiving incoming calls, even if we
        don't want to handle them. }
    procedure ModemConfigDialog;
      { Executes the vendor-supplied modem configuration dialog (the owner
        window will be the application's main form). The changes will not be
        stored permanently (this component does not supply temporary changes
        made by the apllication, it will use the modem as it is configured).
        If the device is not a data-modem then LINEERR_OPERATIONUNAVAIL error
        will occure. }
    procedure Open;
      { Gets a handle for the line. This application wants to own inbound
        data-modem calls on the line. You should not call this method, if line
        is already Active. Changing Active to True will call Open. }
    property Active: Boolean;
      { Line is open. Changing the property's value will call
        Open or Close. Remains unchanged if an exception occurs. }
    property ApiVersion: Cardinal;
      { Negotiated API version agreed by TAPI and SP. Extensions are not
        supported. }
    property DeviceClasses: TStrings;
      { Supported device classes. Available from TAPI 2.0 }
    property Handle: HLINEAPP;
      { The handle representing the opened line device for TAPI.
        If not Active then 0. }
    property ID: Cardinal;
      { Line device identifier. Both ID and Name are unique identifiers. }
    property MediaModes: Cardinal;
      { swMediaModes member of the LINEDEVCAPS structure for the line device. }
    property Name: string;
      { Line device name. Both ID and Name are unique identifiers. }
    property Service: TTapiLineService;
      { The active service that created this object. }
    property OnCallConnected: TCallConnectedEvent;
      { The two data-modems connected succesfully. CommHandle is the handle,
        CommName is the name of open comm. port (e.g. 'COM1'). The port
        is opened using the FILE_FLAG_OVERLAPPED value and must be closed
        by using the CloseHandle function before calling DropCall.
        If the ErrorMsg string is not empty then it is an error message,
        CommHandle and CommName may be invalid. }
    property OnCallDisconnected: TNotifyEvent;
      { If disconnection is not initiated by DropCall then this event handler
        is called. The comm. port handle became invalid, line should be dropped. }
    property OnCallError: TCallErrorEvent;
      { AnswerCall, DropCall or MakeCall completed with an error. Code is one
        of the LINEERR_ constants (may be converted to string by calling
        tapiErrorToStr). Upon an AnswerCall or MakeCall error, OnCallFinished
        will also be called. }
    property OnCallFinished: TNotifyEvent;
      { The call has been finished (even if it could not connect yet to the
        remote station , or if it was an unhandled incoming call), you may start
        a new call by calling MakeCall.
        If dialing started, then the reason of disconnection was passed
        in a pervious OnCallState call. }
    property OnCallID: TCallerIDEvent;
      { The CallerID of the current call has changed. It is typical for
        incoming calls that the ID string is reported by the modem only after
        the first ring, so OnCallOffering will not tell you the caller,
        but it will be reported soon by this event handler. }
    property OnCallOffering: TCallerIDEvent;
      { An incoming call is offering. The call state was already passed in
        a pervious OnCallState call. Outgoing calls are not allowed until this
        call exists. If OnCallOffering event handler has been established,
        then the application is responsible for answering the incoming call
        (or it may just wait until the caller gives up, in which case the
        call will cease by itself).
        CallerID is the caller's address, if empty, then CallerID is not
        available. Flag is one of the LINECALLPARTYID_ constants, it should be
        LINECALLPARTYID_ADDRESS if CallerID is not empty. You may convert flag
```

```
        values to string by tapiCallPartyIDFlagToStr.
        If CallerID is empty you may still want to wait for several seconds
        (for example if Flag is LINECALLPARTYID_UNKNOWN) because CallerID may
        be passed in a following OnCallID event. }
    property OnCallState: TCallStateEvent;
      { The call initiated by MakeCall or reported by OnCallOffering may be
        tracked by this event handler. State is one of the LINECALLSTATE_
        constants, Info is a call-state-dependent information, Priv is the
        changed privilege of the application for the call (may be converted
        to string by tapiCallStateToStr).
        LINECALLSTATE_IDLE state will not be passed, OnCallFinished will
        be called instead. }
    property OnClose: TNotifyEvent;
      { The line device has been forcibly closed. }
  end;

{ TTapiLineService is the invisible component of the TAPI modem services. }
  TTapiLineService = class(TComponent)
  public
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
      { Deactivates TAPI. }
    procedure InitializeEx;
      { Initializes TAPI, enumerates line devices, sets Active property to True.
        You should not call this method, if service is already Active.
        Changing Active to True will call InitializeEx. }
    procedure Shutdown;
      { Shuts down TAPI, frees line devices, sets Active property to False.
        You should not call this method, if service is not Active.
        Changing Active to False will call Shutdown. }
    property Active: Boolean;
      { TAPI is initialized. Changing the property's value will call
        InitializeEx or Shutdown. Remains unchanged if an exception occurs. }
    property ApiVersion: Cardinal;
      { Highest API version supported by TAPI. }
    property Device[ID: Integer]: TTapiLineDevice;
      { Available line devices. Index goes from 0 to DeviceCount - 1. }
    property DeviceByName[const Name: string]: TTapiLineDevice;
      { By the means of this property you may find a device by it's name
        (may be nil, if there is no such device). }
    property DeviceCount: Integer;
      { Number of line devices available. If not Active then 0. }
    property Handle: HLINEAPP;
      { Application's usage handle for TAPI. If not Active then 0. }
  published
    property AppName: string;
      { User friendly name of the application (default = Owner.Caption). }
    property OnMessage: TLineMsgEvent;
      { Line message event handler. This message handler is intended only for
        logging / debugging. Every necessary message handling is carried out
        by the object(s). This event handler will be called first. }
  end;

function HiWord(Value: Longword): Word;
function LoWord(Value: Longword): Word;
function tapiVersionToStr(Version: Cardinal): string;
procedure tapiCheckResult(Code: Integer);
function tapiErrorToStr(Code: Cardinal): string;
function tapiCallPartyIDFlagToStr(Flag: Cardinal): string;
function tapiCallStateToStr(State, Info, Priv: Cardinal): string;
function FormatLineMessage(hDevice, dwMsg, dwCallbackInstance,
  dwParam1, dwParam2, dwParam3: Cardinal): string;

procedure Register;
  { TTapiLineService component will appear on the Others palette page. }

implementation
```

```
end .
```