Boomerang GSM Tutorial

Editor:
Tomáš Mandys, tomas.mandys@2p.cz (2p plus)

Home site:
http://www.2p.cz

Document status:
Version 1.0 First release

Table of Contents

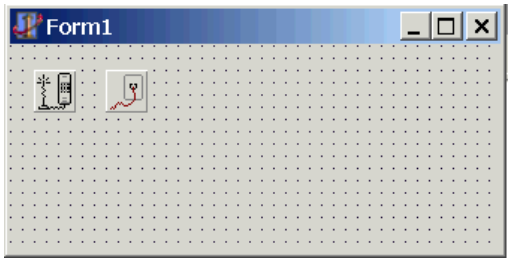
Disclaimer

1. Introduction

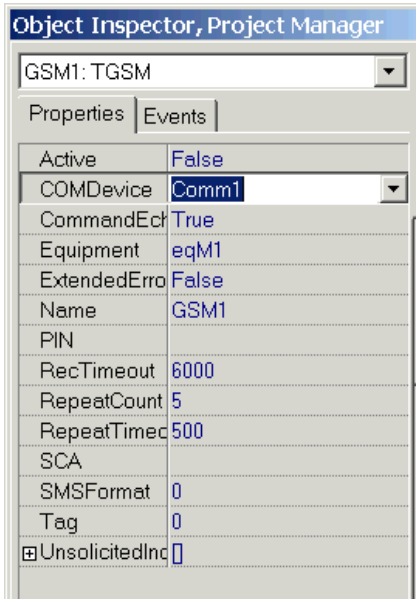
This document describes how to create a simple application based on Boomerang library for SMS communication. GSM specifications are published at ETSI (European Telecommunications Standards Institute – *http://www.etsi.org*) in GSM 3.38 and 3.40 standards (ETSI TS 100 900/901).

2. Basic functionality

Create new application and place *TGSM* and *TComm* components on a form. Both are located probably on *Communication* palette.



Assign Comm1 as GSM1.COMDevice



TGSM component must be active to work. Call *GSM1.Open* method or set *GSM1.Active=True*. Component establishes communication with a GSM module.

```
procedure TForm1.FormShow(Sender: TObject);
begin
  GSM1.Open;
end;
```

GSM component requires some mandatory parameters. Recommended places for setup are in *GSM1.OnBeforeOpen* and *Comm1.BeforeOpen* events. You should know which serial port is the GSM module connected to, GSM operator parameters and SIM card security code (PIN).

```
procedure TForm1.GSM1BeforeOpen(DataSet: TConnection);
begin
  with DataSet as TGSM do
  begin
    Equipment:= eqM35;
    PIN:= '1234';           // or empty string if PIN is not required
    SCA:= '+420602909909';  // phone number of operator service center
    SMSFormat:= smsfPDU;    // or smsfText according to GSM module
  end;
end;
```

```
procedure TForm1.Comm1BeforeOpen(DataSet: TConnection);
begin
  with DataSet as TComm do
  begin
    DeviceName:= 'Com1';
    BaudRate:= br19200;
  end;
end;
```

GSM components are ready for communication. You setup parameters in design time using object inspector, of course.

2.1. Submit message

Create a button and in *OnClick* event do SMS sending functionality. First it's necessary to define what to send via *TSMSSubmit* class instance and then send it using *GSM1.SendSMS* method.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  SMS: TSMSSubmit;
begin
  SMS:= TSMSSubmit.Create;
  try
    SMS.GSM:= GSM1;
    SMS.DA:= '+420604690589';   // destination address
    SMS.UD:= 'My first SMS send using Boomerang library';  // text of message
    GSM1.SendSMS(SMS);
  finally
    SMS.Free;   // destroy object instance
  end;
end;
```

2.2. Deliver message

Now the application can send simple text message and we implement delivery message functionality.

First after *GSM1* component is open then the GSM module will be checked if messages are stored in memory.

```
procedure TForm1.GSM1AfterOpen(DataSet: TConnection);
var
  Sg: TStrings;
  I, J: Integer;
begin
  // retrieve messages from module memory
  if (DataSet as TGSM).Equipment in [eqM20, eqNokia9110, eqWavecom, eqFasttrack] then
    Sg:= GSM1.GetSMSList(4{all})
  else
    Sg:= GSM1.GetSMSList(-1);
  try
    for I:= 0 to Sg.Count-1 do  // process all retrieved messages
    begin
      J:= StrToInt(Sg.Names[I]);
      if StrToInt(Sg.Values[Sg.Names[I]]) in [0{unreaed}] then
      begin
        if Sg.Objects[I] <> nil then
```

```
          ProcessSMS(Sg.Objects[I] as TSMS);
      end;
      GSM1.DeleteSMS(J);
    end;
  finally
    Sg.Free;
  end;
end;
```

There are two possibilities how to process delivered messages. The application can periodically check the GSM module if a message has been delivered using *GSM1.GetSMSList* method or take advantage that the GSM module alerts application that a message has been delivered using *GSM1.OnUnsolicitedSMS* event. But alerting, i.e. unsolicited indication, must be enabled using *GSM1.UnsolicitedIndication* property – add it in *GSM1.OnBeforeOpen* event.

Note that some mobile phones (e.g. Siemens S55) do not alert application if message is delivered unless *uindOnlyIndication* feature is enabled.

```
procedure TForm1.GSM1BeforeOpen(DataSet: TConnection);
begin
  with DataSet as TGSM do
  begin
    ...
    UnsolicitedIndication:= [uindSMSDeliver, uindOnlyIndication];
  end;
end;
```

To the OnUnsolicitedSMS are passed two important parameters, *Idx* and *aSMS*. *Idx* is byte identifier of message in GSM module memory and *aSMS* is instance carrying delivered message data. If *aSMS* is *nil* it's necessary to read it explicitly from GSM module memory using *GSM1.ReadSMS* method.

```
procedure TForm1.GSM1UnsolicitedSMS(Sender: TObject; Idx: Integer;
  aSMS: TSMS);
var
  Stat: Integer;
begin
  if aSMS = nil then
    begin
      try
        aSMS:= GSM1.ReadSMS(Idx, Stat);
        if Stat in [0{unread}] then
          ProcessSMS(aSMS);  // process only unreaded messages
      finally
        GSM1.DeleteSMS(Idx);  // delete message from module memory
      end;
    end
  else
    begin
      ProcessSMS(aSMS);
      if Idx <> -1 then
        GSM1.DeleteSMS(Idx);
    end;
end;
```

Message is passed to *ProcessSMS* where message is processed. It's necessary to test type of *aSMS* instance because more message types are delivered by this manner. Delivered message is defined as *TSMSDeliver* class.

```
procedure TForm1.ProcessSMS(aSMS: TSMS);
var
  S: string;
begin
  if aSMS is TSMSDeliver then
    begin
      with aSMS as TSMSDeliver do
        S:= Format('>%s, OA: %s, UD: %s',
             [DateTimeToStr(SCTS), OA, UD]);
    end
  else
    Exit;
  MessageDlg(S, mtInformation, [mbOk], 0);
end;
```

Now is the application able to send and show delivered messages.

2.3. Status report

GSM network implements notification if message was delivered to destination address, i.e. status report. Status report request is send with SMS. Just add one line to *Button1Click* event.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  ...
    SMS.SRR:= True;  // status report request
    GSM1.SendSMS(SMS);
  ...
end;
```

GSM operator sends the status report to originator as special message. GSM module alerts application that a message has been delivered using *GSM1.OnUnsolicitedSMS* event, enable alerting using *GSM1.UnsolicitedIndication*

```
procedure TForm1.GSM1BeforeOpen(DataSet: TConnection);
begin
  with DataSet as TGSM do
  begin
    ...
    UnsolicitedIndication:= [uindSMSDeliver, uindSMSStatusReport,
                             uindOnlyIndication];
  end;
end;
```

Add processing algorithm to *ProcessSMS* method as new fork. The status report is defined as *TSMSStatusReport* class.

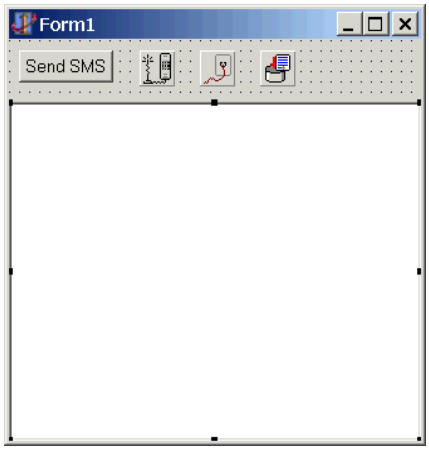
```
procedure TForm1.ProcessSMS(aSMS: TSMS);
var
  S: string;
begin
  ...
  else if aSMS is TSMSStatusReport then
```

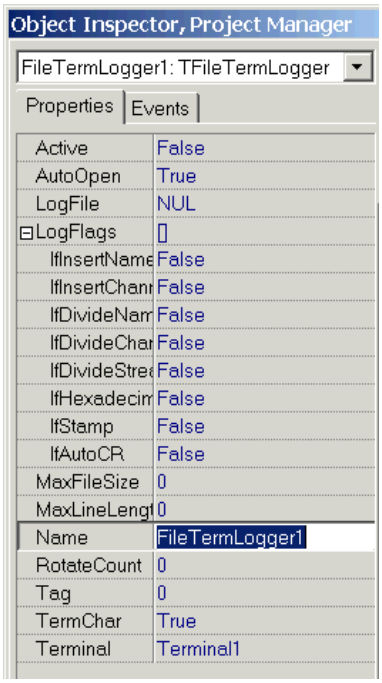
```
    begin
      with aSMS as TSMSStatusReport do
        S:= Format('!%s, OA: %s, Status: %d, DT: %s',
            [DateTimeToStr(Now), OA, Status, DateTimeToStr(DT)]);
    end
  ...
end;
```

Note that not all modules support status report.

2.4. Serial communication

Serial communication between machine and GSM module can be logged for custom purposes. Let's show how to log it in terminal window. Simply add a *TTerminal* visible VCL component and *TFileTermLogger* non-visual component. Both components should be located on *Communication* palette. Connect *Comm1* to *FileTermLogger1* and *Terminal1* to *FileTermLogger1*. All characters flowing through serial port will be passed to *FileTermLogger1* component and then to visual terminal window.



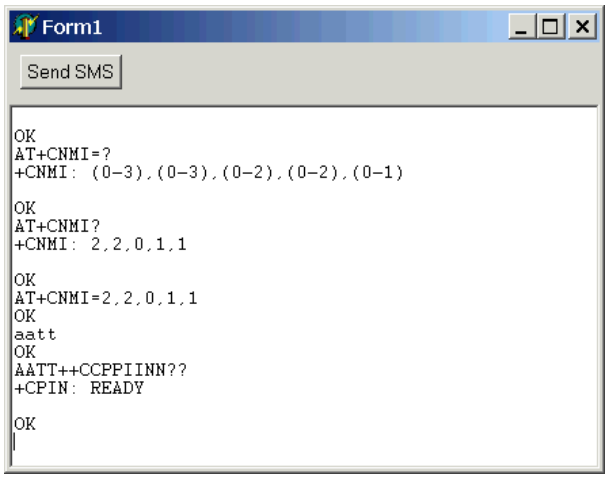


To enable user input in terminal perform following action in *Terminal1.OnKeyPressEvent*.

```
procedure TForm1.Terminal1KeyPress(Sender: TObject; var Key: Char);
begin
  if not GSM1.Active then
    GSM1.COMDevice.Close;   // reopen to take effect potential new Comm1 parameters

  GSM1.COMDevice.Open;
  GSM1.COMDevice.Send(Key); // send key to GSM module
  Key:= #0;
end;
```

Note that all characters are echoed back then will be logged to terminal window twice.



2.5. Character set

Message text in standard message is encoded in 7-bit set then codes between #0 and #127 are possible. *GSM* unit provides two functions to transliterate between ANSI and GSM code page. Alphabet characters have the same code in both sets but e.g. “at” (“@”) sign has in GSM code page assigned code *#0*. Maximal length of 7-bit message is 160 characters.

```
S:= ISOtoSMS('myname@mydomain.com');

S:= SMStoISO(aSMS.UD);
```

GSM standard provides also 8-bit messages, i.e. binary messages. Maximal length of binary message is 140 characters. To send binary message set *TSMSSubmit.DCS* property value to *$F5* (or *$04*).

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  ...
    SMS.DCS:= $04;  // or $16, $F5 // binary 8-bit message
    GSM1.SendSMS(SMS);
  ...
end;
```

Next supported code page is 16-bit UCS-2, i.e. Unicode. Maximal length of Unicode message is 70 characters. To send Unicode message set *TSMSSubmit.DCS* property value to *$08* (or *$1A*). Feed message text to *TSMSSubmit.WideUD* instead to *TSMSSubmit.UD* property.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  ...
    SMS.DCS:= $08;  // or $1A // Unicode message
    SMS.WideUD:= 'My first Unicode SMS send using Boomerang';  // text of message
    GSM1.SendSMS(SMS);
  ...
end;
```

See documentation of *DCS* field. Note that some modules do not support all standardized *DCS* codes. For example Siemens M20 does not support $08, $04 anyway and $16, $1A with bug. It depends also on GSM module firmware.

2.6. Message classes

Message class defines target message platform. There are four message classes – class 0 to 3. The most of mobile phones support message class 0 and 2.

Class 0	display immediately	
Class 1	ME (mobile equipment)	
Class 2	SIM	
Class 3	TE (terminal equipment)	

Message class is determined using *TGSM.DCS* property. If *DCS* is set to *0* default class is used (probably Class 2)

```
procedure TForm1.Button1Click(Sender: TObject);
  ...
    SMS.DCS:= $F0;   // message class 0
    SMS.UD:= 'My first message to display immediately';
    GSM1.SendSMS(SMS);
  ...
end;
```

See documentation of *DCS* field.

2.7. Blinking (flash) messages

On some phones (all Nokias, some Siemens, Ericsson, Motorola etc.) a class 0 message will appear as a flash SMS message. These messages appear on the screen immediately upon arrival, without the need to press any buttons on the phone. If the data coding scheme is set to 16-bit UCS-2 (Unicode), and the message starts with character #1, it will appear as a blinking flash message. Some phones provides “enable/disable” blinking tag characters that enable blinking of part of message. Tag characters are defined in *XSMS.pas* as *smBlinkOn* and *smBlinkOff* constants.

```
procedure TForm1.Button1Click(Sender: TObject);
  ...
    SMS.DCS:= $18;   // UNICODE message, class 0
    SMS.WideUD:= #1+'BLINK';
    GSM1.SendSMS(SMS);
  ...
end;
```

3. Smart Messages®

Smart Messages were introduced by Nokia and are supported on Nokia phoned. It enables transmitting of binary content (logo, ringing tone, vCard, vCalendar) over the air in using plain SMS. Binary content may be larger than 140/160 character then does not fit to one message.

There is defined protocol that enables splitting of content at originating side to more messages i.e. packets and concatenating them at target side. Protocols are implemented in *XSMS.pas* unit. There are abstract protocol classes – *TSMSProtocol*, *TSMSProtocol2* – and concrete *TNBS* and *TUDHProtocol* classes. Note that Smart Messages except vCards and vCalendars use UDH (User Data Header) then *TSMSSubmit.UDHI* property must be set to *True* and *DCS*

as binary 8-bit messages. Content is identified by a defined port number (see *smPortxxxx* constants).

Binary content is defined using *TSmartMessage* class.

Note that color pictures and polyphony melodies may not be transmitted via Smart Messages.

Detailed specifications are published at Nokia developer site *http://www.forum.nokia.com*.

```
procedure TForm1.SendSmartMessage(SM: TSmartMessage; aDestAddress: Word);
var
  I: Integer;
  NBS: TSMSProtocol2;
  SMS: TSMSSubmit;
begin
  SMS:= TSMSSubmit.Create;
  try
    SMS.GSM:= GSM1;
    SMS.DA:= DestAddress;   // destination address

    if SM.InheritsFrom(TSMMIMEDirectory) then
      begin
        SMS.DCS:= 0;        // default message class
        NBS:= TNBS.Create   // vCard and vCalendar
      end
    else
      begin
        SMS.DCS:= $F6;         // binary message, class 2, SIM
        SMS.UDHI:= True;       // user data header indicator
        NBS:= TUDHProtocol.Create;
      end;
    try
      NBS.Options:= [smspoAddressing, smspoReference];
      NBS.DestinationAddress:= aDestAddress;  // set port address
      NBS.SourceAddress:= NBS.DestinationAddress;
      NBS.Data:= SM.Data;      // feed binary data

      NBS.Reference:= fReference mod 256;  // unique identifier
      Inc(fReference);         // every message has unique identifier
      for I:= 1 to NBS.FragmentCount do   // send particular packets
      begin
        SMS.UD:= NBS.Fragments[I];
        GSM1.SendSMS(SMS);    // send message
      end;
    finally
      NBS.Free;
    end;
  finally
    SMS.Free;
  end;
end;
```

3.1. vCard

The simplest Smart Message type is vCard carrying personal contact data defined in *TSMvCard* class.

```
procedure TForm1.Button2Click(Sender: TObject);
```

```
var
  SM: TSMvCard;
begin
  SM:= TSMvCard.Create;
  try
    SM.Records.Values['N']:= 'My test contact'; // name
    SM.Records.Values['TEL']:= '+420604123456'; // phone number
    SendSmartMessage(SM, smPortMIMEvCard);
  finally
    SM.Free;
  end;
end;
```

3.2. Logo

Logo is a monochrome picture that appears on mobile display. Mobile logo is assigned to operator which network mobile is logged in hence the operator must be identified via *MNC* and *MCC* code. Picture dimension depends on mobile phone. Boomerang implements logo in *TSMOperatorLogo* class. To delete operator logo set *MNC* and *MCC* to zero.

```
procedure TForm1.Button3Click(Sender: TObject);
var
  SM: TSMOperatorLogo;
  Bmp: TBitmap;
const
  OperatorID= 23001;  // T-Mobile Czech
begin
  SM:= TSMOperatorLogo.Create;
  try
    SM.Width:= 16;                // picture dimension
    SM.Height:= 16;
    SM.MCC:= OperatorID div 100;  // GSM operator network identifier
    SM.MNC:= OperatorID mod 100;
    Bmp:= TBitmap.Create;
    try
      Bmp.LoadFromFile('logo.bmp');  // bitmap mask
      SM.ImportFromImage(Bmp);
    finally
      Bmp.Free;
    end;
    SendSmartMessage(SM, smPortOperatorLogo);
  finally
    SM.Free;
  end;
end;
```

Note that is defined also CLI (Calling Line Identification) icon that may assigned to a phone list item (*TSMCLIIcon* class).

3.3. Ringing tone / melody

Ringing tone functionality is implemented in *TSMRingingTone* class. Only simple one instrument melodies may be transmitted. Library support simple import and export to *RTTTL* format. There are tools that support conversion from a MIDI file to RTTTL format.

```
procedure TForm1.Button4Click(Sender: TObject);
var
```

```
  SM: TSMRingingTone;
  St: TStream;
begin
  SM:= TSMRingingTone.Create;
  try
    St:= TFileStream.Create('melody.rtt', fmOpenRead);
    try
      SM.RTTTL:= StreamToString(St);  // load from file in RTTTL format
    finally
      St.Free;
    end;
    SendSmartMessage(SM, smPortRingingTone);
  finally
    SM.Free;
  end;
end;
```

3.4. Picture message

Smart Messages support picture messages that enable combining picture and text into one message. There is defined *TSMMultipartMessage* class that behaves like text and picture container.

```
procedure TForm1.Button5Click(Sender: TObject);
var
  SMMP: TSMMultipartMessage;
  SM: TSMOTABitmap;
  SMT: TSMTextISO;
  Bmp: TBitmap;
begin
  SMMP:= TSMMultipartMessage.Create;
  try
    SM:= TSMOTABitmap.Create;     // picture
    SM.Width:= 16;                // picture dimension
    SM.Height:= 16;
    Bmp:= TBitmap.Create;
    try
      Bmp.LoadFromFile('logo.bmp');  // bitmap mask
      SM.ImportFromImage(Bmp);
    finally
      Bmp.Free;
    end;
    SMMP.Multiparts.Add(SM);

    SMT:= TSMTextISO.Create;      // additional text
    SMT.Text:= 'My first commented picture message';
    SMMP.Multiparts.Insert(0, SMT); // add above picture

    SendSmartMessage(SMMP, smPortMultiPart);
  finally
    SMMP.Free;
  end;
end;
```

4. EMS

EMS standard is supported in Siemens, Ericsson, Alcatel phones. It enables transmitting of binary content (logo, ringing tone, vCard, vCalendar) over the air in using plain SMS. Binary

objects are placed into text. Packets of EMS messages are transmitted with UDH (User Data Header) as binary messages. *TEMSUDHProtocol* class is used and support dividing of long objects using UPI (User Prompt Indicator).

Binary content is defined using *TEMS* container class. The container contains any number of *TEMSObject objects.*

Note that even EMS specification defines color pictures, polyphony melodies and digitized sounds but it's not supported in phones anyway. Upload of such objects is performed only using WAP Push or MMS technology.

Specification may be found at *http://www.3gpp.org* in *"3GPP TS 23.040"* standard.

```
procedure TForm1.SendEMS(EMSObj: TEMSObject; aText: string);
var
  I: Integer;
  NBS: TUDHProtocol;
  EMS: TEMS;
  SMS: TSMSSubmit;
begin
  SMS:= TSMSSubmit.Create;
  try
    SMS.GSM:= GSM1;
    SMS.DA:= DestAddress;   // destination address
    SMS.DCS:= $F6;         // binary message, class 2, SIM
    SMS.UDHI:= True;       // user data header indicator
    NBS:= TEMSUDHProtocol.Create;
    try
      NBS.Options:= [smspoReference];
      NBS.Reference:= fReference mod 256;  // unique identifier
      Inc(fReference);         // every message has unique identifier
      EMS:= TEMS.Create;
      try
        EMS.Text:= aText;
        EMS.Objects.Add(EMSObj);  // add object to EMS container
        EMS.WriteTo(NBS);      // write data to protocol object
      finally
        EMS.Free;
      end;

      for I:= 1 to NBS.FragmentCount do   // send particular packets
      begin
        SMS.UD:= NBS.Fragments[I];
        GSM1.SendSMS(SMS);    // send message
      end;
    finally
      NBS.Free;
    end;
  finally
    SMS.Free;
  end;
end;
```

4.1. Picture

Picture is a monochrome mask that appears in message. Phone operator can save message e.g. as logo directly at phone. Library implements EMS pictures in *TEMSPictureSmall* (16x16 pixels), *TEMSPictureLarge* (32x32 pixels) and *TEMSPictureVariable* classes.

```
procedure TForm1.Button6Click(Sender: TObject);
var
  EMS: TEMSPicture;
  Bmp: TBitmap;
begin
  EMS:= TEMSPictureSmall.Create;  // 16x16 pixels
  Bmp:= TBitmap.Create;
  try
    Bmp.LoadFromFile('logo.bmp');  // bitmap mask
    EMS.ImportFromImage(Bmp);
  finally
    Bmp.Free;
  end;
  SendEMS(EMS, 'My first EMS small picture (16x16) pixels');
end;
```

4.2. Animation

Animation is set of four monochrome masks that are periodically displayed in message. Library implements custom EMS pictures in *TEMSAnimationSmall* (8x8 pixels) and *TEMSAnimationLarge* (16x16 pixels) classes and predefined animation in *TEMSAnimationPredef* class (see *emsanixxxx* constants)

```
procedure TForm1.Button7Click(Sender: TObject);
var
  EMS: TEMSAnimation;
  Bmp: TBitmap;
begin
  EMS:= TEMSAnimationLarge.Create;  // 16x16 pixels
  Bmp:= TBitmap.Create;
  try
    Bmp.LoadFromFile('anim.bmp');  // bitmap mask 32x8 pixels
    EMS.ImportFromImage(Bmp);
  finally
    Bmp.Free;
  end;
  SendEMS(EMS, 'My first EMS large animation (16x16) pixels');
end;
```

4.3. Sound

Custom EMS sound functionality is implemented in *TEMSSoundUserDef* class. Only simple one instrument melodies are supported. Library supports import from *RTTTL* format using *TSMRingingTone*. Custom sounds are implemented in *TEMSSoundPredef* class (see *emssndxxxx* constants).

```
procedure TForm1.Button8Click(Sender: TObject);
var
  EMS: TEMSSoundUserDef;
  SM: TSMRingingTone;
  St: TStream;
begin
  EMS:= TEMSSoundUserDef.Create;
```

```
  SM:= TSMRingingTone.Create;
  try
    St:= TFileStream.Create('melody.rtt', fmOpenRead);
    try
      SM.RTTTL:= StreamToString(St);  // load from file in RTTTL format
      EMS.ImportFromSM(SM);
    finally
      St.Free;
    end;
  finally
    SM.Free;
  end;
  SendEMS(EMS, 'My first EMS sound');
end;
```

5. Siemens Exchange Object (SEO)

Starting with the S 45 and ME 45, Siemens mobile phones provide over-the-air (OTA) download capabilities for different types of content – Windows bitmaps and monophonic MIDI ring tones.

Data are divided into packets and concatenated using *TSiemensOTA* protocol class.

See "*Siemens Mobile Phones: Specifications for OTA-Download Service"* for more information.

```
procedure TForm1.SendSEO(aFileName: string);
var
  NBS: TSiemensOTA;
  St: TStream;
  SMS: TSMSSubmit;
  I: Integer;
begin
  SMS:= TSMSSubmit.Create;
  try
    SMS.GSM:= GSM1;
    SMS.DA:= DestAddress;   // destination address
    SMS.DCS:= $F5;          // binary message, class 1
    NBS:= TSiemensOTA.Create;
    try
      NBS.Reference:= fReference mod 256;   // unique id
      Inc(fReference);
      NBS.ObjectName:= ExtractFileName(aFileName);
      NBS.ObjectType:= Copy(LowerCase(ExtractFileExt(aFileName)), 2, 3);// bmp or mid
      St:= TFileStream.Create(aFileName, fmOpenRead);
      try
        NBS.Data:= StreamToString(St);   // feed binary data
      finally
        St.Free;
      end;
      for I:= 1 to NBS.FragmentCount do   // send packets
      begin
        SMS.UD:= NBS.Fragments[I];
        GSM1.SendSMS(SMS);    // send message
      end;
    finally
      NBS.Free;
    end;
  finally
    SMS.Free;
  end;
end;
```

6. WAP Push

Color pictures and polyphonic melodies may be transmitted using WAP Push technology. Mobile phone must support WAP. Individual obtains special message (WAP push SMS) containing link to a resource at public network place. Mobile phone recognized it and according type provides resource download on demand. Individual manually confirms that want download resource in WAP browser. Hence resource is downloaded not as message but as data using GPRS or CSD dial-up.

WAP Push message is created using *TUDHProtocol* and *TWAPService* classes. Resource is determined as WBXML document. WBXML format is binary equivalent of WML text format. WBXML is compressed to save limited bandwidth. Data of WAP document are stored in *TWapDocument* class instance.

There are three supported WAP Push services in the library – *TWapServiceIndication*, *TWapServiceLoad* and *TWapCacheOperation* classes.

Specification may be found at *http://www.wapforum.org* and *http://www.openmobilealliance.org*.

```
procedure TForm1.SendWapPush(aWapPushObj: TWapService; aType: Byte);
var
  NBS: TUDHProtocol;
  SMS: TSMSSubmit;
  I: Integer;
  WD: TWapDocument;
begin
  SMS:= TSMSSubmit.Create;
  try
    SMS.GSM:= GSM1;
    SMS.DA:= DestAddress;   // destination address
    SMS.DCS:= $F6;          // binary message, class 2, SIM
    SMS.UDHI:= True;
    NBS:= TUDHProtocol.Create;
    try
      NBS.Options:= [{smspoReference, }smspoAddressing];
      NBS.DestinationAddress:= smPortWAPPushConnectionlessSessionService_client;
      NBS.SourceAddress:= smPortWAPConnectionlessSessionService;
      WD:= TWapDocument.Create;
      try
        aWapPushObj.AssignToWapDocument(WD);   // encode as WAP document
        NBS.Data:= #01#06#01+Chr(aType or $80)+WD.WBXMLData;  // magic header
      finally
        WD.Free;
      end;

      NBS.Reference:= fReference mod 256;
      Inc(fReference);

      for I:= 1 to NBS.FragmentCount do   // send particular packets
      begin
        SMS.UD:= NBS.Fragments[I];
        GSM1.SendSMS(SMS);                // send message
      end;
    finally
      NBS.Free;
    end;
  finally
    SMS.Free;
```

```
  end;
end;

procedure TForm1.Button11Click(Sender: TObject);
var
  WP: TWapServiceIndication;
begin
  WP:= TWapServiceIndication.Create;
  try
    WP.HRef:= 'http://www.2p.cz/wap/example/test.jpg';  // an object address
    WP.Caption:= 'My first launching bird';  // description that appear in SMS
    SendWapPush(WP, $2E);
  finally
    WP.Free;
  end;
end;
```

7. Advanced delivery processing

Large messages which content does not fit into one SMS are divided at originating side to more packets. Library supports concatenating of Smart Messages, EMS and Siemens Exchange Objects. All delivered messages are passed through *TSMSProtocolStack* class that tries to group corresponding packets. Does not matter in which order the packets are coming. The instance of *TSMSProtocolStack* class must be persistent hence is declared in private part of the form.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  FSMSProtocol:= TSMSProtocolStack.Create;
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  FSMSProtocol.Free;
end;
```

ProcessSMS method has been enhanced of passing messages to *fSMSProtocol* object. First expired non-complete messages are removed in *fSMSProtocol.CleanSMSProtocols* method. If message is not completed in a few tens of seconds it won't be completed probably anymore – some message packet was lost. In the example are cleaned messages non-completed to 5 minutes.

Next delivered message (packet) is processed in *fSMSProtocol.ProcessSMS* method. If method recognizes that message is a packet then returns *True* and *NBS* object of *TSMSProtocol*. *NBS.Status* property holds status – if message is completed or not. According *NBS* class type is type of message is recognized. When concatenated message is processed then is removed from the stack using *fSMSProtocol.RemoveSMSProtocol*.

```
procedure TForm1.ProcessSMS(aSMS: TSMS);
var
  S, S2: string;
  I: Integer;
  NBS: TSMSProtocol;
  Addr: Word;
  SM: TSmartMessage;
```

```
  EMS: TEMS;
begin
  if aSMS is TSMSDeliver then
    begin
      with aSMS as TSMSDeliver do
    ...

      // do advanced processing
      // remove expired messages
      fSMSProtocol.CleanSMSProtocols(5{min}/60/24, False);

      // check concatanation status of delivered message
      if fSMSProtocol.ProcessSMS(aSMS, NBS) then
      begin      // message is concatable
        Addr:= 0;
        if NBS is TSMSProtocol2 then
          Addr:= TSMSProtocol2(NBS).DestinationAddress;
        S2:= Format('*ref:%.2x, #%d/%d, , addr:%.4x',
                    [NBS.Reference, NBS.InsertedCount, NBS.FragmentCount, Addr]);
        if NBS.Status = smspsOK then
        begin      // message is last part
          S2:= S2+' complete';
          { here is possible process received data, show bitmap, play melody,
            save it to file etc. }
          if NBS is TSMSProtocol2 then
            begin
              // first check if message is Smart Message
              SM:= TSmartMessage.CreateSM(NBS.Data,
                                          TSMSProtocol2(NBS).DestinationAddress);
              if SM <> nil then
                try
                  S2:= S2 + ' SmartMessage:'+SM.ClassName;
                  {if SM is TSMMultipartMessage then
                    S2:= S2+'Multipart message'
                  else if SM is TSMRingingTine then
                    S2:= S2+'Ringing tone'  classname  inheritsfrom }
                finally
                  SM.Free;
                end
              else if NBS is TUDHProtocol then
                begin
                  // if it's not Smart Message it's EMS
                  EMS:= TEMS.Create;
                  try
                    try
                      EMS.ReadFrom(TUDHProtocol(NBS));
                      S2:= S2+ ' EMS';
                      for I:= 0 to EMS.Objects.Count-1 do
                      begin
                        if I > 0 then
                          S2:= S2+',';
                        S2:= S2+EMS.Objects[I].ClassName;
                      end;
                    except
                    end;
                  finally
                    EMS.Free;
                  end;
                end;
            end
          else if NBS is TSiemensOTA then
```

```
            begin
              S2:= S2 + Format(' SEO: %s, Name: "%s"',
                               [string(TSiemensOTA(NBS).ObjectType),
                                string(TSiemensOTA(NBS).ObjectName)]);
            end;

          fSMSProtocol.RemoveSMSProtocol(NBS);  // remove message from stack
          NBS.Free;
        end;
        S:= S+#13#10+S2;
      end;
    end
  ...
end;
```