Boomerang Library Guide

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

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

Product overview

BOOMERANG is the component library for GSM communication software development. The components are primary written for Delphi 5, but they are usable also for Delphi 4,6,7 and Kylix 3 and finally for Delphi 8 for .NET Framework.
Some components are available as ActiveX objects usable by any ActiveX capable application (Visual Basic, Visual C#, Visual Basic for Applications, Delphi etc.).

Features

- (serial) port components
- modem components (client dial-up communication)
- socket components (TCP/IP client side)
- TAPI components (TAPI 2.0 - Win98, WinNT)
- GSM modem components
- BMG components (Business Message Gateway), Nokia CIMD protocol(Computer Interface to Message Distribution)
- BMG client
- EMI components (ERMES UCP, T-Online SMS Direct, Oskar OSKO)
- EMI client
- terminal component
- narrow band socket (NBS) protocol, UDH (user data header) protocol
- OTA (over-the-air) services
- Smart messages (CLI, logos, picture messages, ringing tones, business cards, calendar)
- EMS (pictures, animations, sounds, etc.)
- OCX libraries for SMS and EMI communication
- Wap Push messages
- WAP WBXML support

The components are multi-threaded. It means, that communication components can work asynchronously (non-modally).

History

Here is brief history of development progress.

July 2004 (v 5.2)

- WBXML support
- Wap-push messages

April 2004

- fixed bug in AT+CNMI for Siemens S55, ...
- tested Bluetooth and IrDA
- tested Nokia 6600

December 2003

- EMIClient and GSM as ActiveX library
- new demo application
- changed license model

November 2003

- added Wavecom Fasttrack support (and Fargo Maestro)
- TGSM.CommandEcho property
- fixed bugs in TSiemensOTA
- fixed bug in Wavecom initialization (wait for +WIND)

October 2003

- import/export of Smart Message ringing tone to RTTTL

September 2003

- extended EMI component (common keep-alive)
- new events RxCommandBeforeAcknowledge for TEMIClient and TBMGClient

August 2003

- Nokia 6650 supported

July 2003 (v. 4.0)

- SMS protocol support (NBS, UDHI, Siemens)
- Smart messages
- EMS messages
- added support for accessing UD in UNICODE (widestring)

July 2003

- fixed some bugs (ocassionally striped characters in PDU)

April 2003 (v. 3.0)

- ported to Kylix
- rewritten UnsolicitedIndication
- get ManufacturerInfo support
- new SMS type support (SMS-STATUS-REPORT, ...)
- cell broadcast support
- help (.hlp) file

June 2002

- fixed bug in TTerminal,TSocketConnection thread initialization (in D6)

February 2002

- bugs in timing of TGSM destroying fixed
- rewritten (accelerated) visual TTerminal

January 2002

- *basic* distribution discontinued, all of them is in *lite* one

December 2001

- new component TModem for dial-up communication and connection

November 2001

- fixed bug in Windows NT/2000 when object being destroyed (added WaitFor thread destroyed)
- fixed mistake in TGSM,TBMGClient,TEMIClient if main thread was not also VCL thread

September 2001

- Siemens TC35 supported

august 2001

- new component TSocketConnection for TCP/IP communication

July 2001

- WaveCom supported
- Nokia 6210 supported
- Siemens M35, Motorola T260 supported
- special kind of phone number, alias support implemented (for ex. 4616)

July 2000

- Nokia 9110 and SMS text format supported

Boomerang applications

There is includes several application founded on library.

- EMI CLIENT (p.6)
- BMG CLIENT (p.7)
- Demos (p.9)

EMI CLIENT

Is a sample fully functional application that demonstrates all the TEMIClient (p.21) functions. Since it is an automation server its function can be used by other application through COM/DCOM technology.
There are enabled two methods of keep-alive. First method uses command 31 (SMT->SMSC) and second one uses specific bidirectional messaging - commands 51 (SMT->SMSC) and 52 (SMSC->SMT) with configurable specification.
Before using a automation server is necessary to register it executing EMI client program with the */regserver* parameter, unregistering then with the */unregserver* parameter.
Received messages are passed to client using events.

Command line

`/I:<file_name>` overrides default ini file, default one is `<program_path>/EMI.INI`

Ini file

Most of parameters is written by EMI client except section [EMI.Connection] where are defined operator specific parameters. See TEMIClient.Params (p.21) description. One exception is that *Password* is scrambled.

OLE interface

```
property Active: WordBool;  (read/write)
```

State of connection, set to True to connect to EMI server or False to disconnect from them. When an OLE client is connected EMIClient session is opened. There is not necessary to write to Active property unless you need implement special behaviour.

```
property PhoneNumber: WideString; (read only)
```

Assigned phone number to EMI service (probably short alias, 4 digits)

```
procedure Submit(const DA: WideString; const Msg: WideString;
                 Binary: WordBool; MCLs: Smallint; PID: Smallint;
                 NT: Smallint; RPI: Smallint; VP: TDateTime;
                 out SCTS: TDateTime);
```

Submits message, returns time stamp (SCTS) assigned to message in SMS center. See *EMI protocol datasheet*

```
function  TestConnection: WordBool;
```
Tests connection and returns True if is still active (EMI server drops connection after defined interval of inactivity)

Events

```
property OnActiveChanged: TNotifyEvent;
```
Fired when is changed state of connection (Active property)

```
property OnDelivered: TAutoEMIOnDelivered;
TAutoEMIOnDelivered = procedure(Sender: TObject; SCTS: TDateTime; var OA: OleVariant;
                                                 var DA: OleVariant;var Msg: OleVariant;
                                                 Binary: WordBool; MCLs: Smallint; PID: Sm
                                                 RPI: Smallint; VP: TDateTime) of object;
```
Fired when is delivered a message. See *EMI protocol datasheet*

```
property OnDeliveredNotification: TAutoEMIOnDeliveredNotification read FOnDeliveredNotific
TAutoEMIOnDeliveredNotification = procedure(Sender: TObject; SCTS: TDateTime; var OA: OleV
                                            var DA: OleVariant;
                                            var Msg: OleVariant;PID: Smallint;
                                            DeliveryStatus: Smallint;
                                            ReasonCode: Smallint;
                                            DSCTS: TDateTime) of object;
```
Fired when is delivered a message delivery notification. See *EMI protocol datasheets*.

See also
BMG CLIENT (p.7) GDEP CLIENT (p.8)

BMG CLIENT

Is a sample fully functional application that demonstrates all the TBMGClient (p.12) functions. Since it is an automation server its function can be used by other application through COM/DCOM technology. Underlying protocol is TCP/IP protocol or dial-up connection using modem AT commands or a TAPI interface.
Before using a automation server is necessary to register it executing BMG client program with the */regserver* parameter, unregistering then with the */unregserver* parameter.
Received messages are passed to client using events or are polled by OLE client.

Command line

`/I:<file_name>` overrides default ini file, default one is `<program_path>/BMG.INI`
`/TAPI` use TAPI instead modem connection

OLE interface

```
property Active: WordBool;  (read only)
```
State of connection, True if connected to BMG Center.

```
property RetrieveAut: WordBool;  (read only)
```

True if is enabled RetrieveAut.

```
procedure NewProfile(const Profile1: WideString; const Number: WideString);
procedure GetProfiles(out aValue: WideString);
procedure CopyProfile(const Profile1: WideString; const Profile2: WideString);
procedure AppendProfile(const Profile1: WideString; const Profile2: WideString);
procedure DeleteProfile(const Profile1: WideString);
procedure TestUser(const Profile1: WideString; const Number: WideString; out aValue: WordB
procedure NewUser(const Profile1: WideString; const Number: WideString);
procedure GetUsers(const Profile1: WideString; out aValue: WideString);
procedure DeleteUser(const Profile1: WideString; const Number: WideString);
procedure CountUser(const Profile1: WideString; out aValue: Integer);
procedure Submit(const Number: WideString; const Msg: WideString; VP: Byte; PID: Byte; DCS
procedure Retrieve(out aValue: WideString);
procedure RetrieveAll(out aValue: WideString);
procedure SubmitProfile(const Profile1: WideString; const Msg: WideString; VP: Byte; PID:
procedure Count(out aValue: Integer);
procedure DeleteAll;
procedure SubmitBin(const Number: WideString; const Msg: WideString; VP: Byte; PID: Byte;
procedure SubmitProfileBin(const Profile1: WideString; const Msg: WideString; VP: Byte; PI
```

See documentation of *BMG protocol*

Events

`property OnActiveChanged: TNotifyEvent;`

Fired when is changed state of connection (Active property)

```
property OnRetrievedAut: TAutoBMGOnRetrievedAut read FOnRetrievedAut write FOnRetrievedAut
TAutoBMGOnRetrievedAut = procedure(Sender: TObject; var aValue: OleVariant) of object;
```

Fired when is delivered a message (if `RetrieveAut = True`).

See also

EMI CLIENT (p.6) GDEP CLIENT (p.8)

GDEP CLIENT

Is a sample fully functional application that demonstrates all the TGDEPClient (p.31) functions. Underlying protocol is HTTP over VPN or public internet (SSL secured).

Command line

`/I:<file_name>` overrides default ini file, default one is `<program_path>/GDEPCLIENT.INI`

See also

EMI CLIENT (p.6) BMG CLIENT (p.7)

Demo applications

Example_COM
The project demonstrates using of the serial communication component - sending, receiving, terminal. Example is availbable for Win32, .NET and Linux.

Example_TAPI
The project demonstrates using of the TAPI components - sending, receiving, terminal, automatic dialup, direct connection, configuration settings.

Example_GSM
The project demonstrates using of the GSM modem components - GSM network login, sending and receiving of SMS messages, signal level, manual modem controlling by AT commands. It also demonstrates sending Smart messages and EMS (logos, melodies, animations, etc.). Example is availbable for Win32, .NET, ActiveX and Linux.
Note in the *gsm.ini* configuration file must be set the modem type, the PIN and the SMS center number. The INI is searched in the same directory as the program. The INI file can be changed using */I:<my ini>*.

Registration

Registration is required for each computer where Boomerang library (p.1) is running. The for free registration/activation is valid only for one machine where developer tool is running. If you need deploy your software, you must buy deployment key.
You can register using online registration form (http://www.2p.cz/en/bumerang/registrace.html). You'll receive activation key immediately to your mailbox.

Unit BMG

BMG (*Business Message Gateway*) is an industrial communication interface used by the *Eurotel Czech and Slovakia* GSM operator for mass SMS message sending and receiving. *BMG* protocol is based on Nokia CIMD/BIP protocol (Computer Interface to Message Distribution). The client is connected to the *BMG* via a dialup or a leased line. The components use connecting through the standard TAPI. Receiving of incoming messages can be both asynchronous (unsolicited) and synchronous (solicited). The *BMG* specification is available in *Eurotel*. The components are tested on *BMG v1.17* server release.

TBMG

```
type
  TBMG = class( TLogConnection );
```

Base class encapsulating BMG functionality

Params (TBMG)

```
published
  property Params: TStrings read write;
```

Parameters necessary for connecting (login name, password, etc.), see BMG documentation.

ComDevice (TBMG)

```
published
  property ComDevice: TCommunicationConnection read write;
```

The device that is used for connecting to the BMG (for ex. TModem (p.16), TClientSocketConnection (p.53), TLineCom (p.56))

RepeatCount (TBMG)

```
published
  property RepeatCount: Integer read write
      default 5;
```

Number of attempts to send command.

Version (TBMG)

```
public
  property Version: Word read;
```

The BMG server version.

OnRxChar (TBMG)

```
published
  property OnRxChar: TRxCharEvent (p.Chyba! Záložka není definována.) read write;
```

Is called in the VCL thread when a char was received.

OnRxCommand (TBMG)

```
published
  property OnRxCommand: TRxCommandEvent (p.Chyba! Záložka není definována.) read write;
```

Is called in the VCL thread when a command was received. See also OnRxCommandBeforeAcknowledge (p.12)

OnRxCommandBeforeAcknowledge (TBMG)

```
published
  property OnRxCommandBeforeAcknowledge: TRxCommandBeforeAcknowledgeEvent (p.Chyba! Záložka není
```

Is called in the COM thread when a command was received. *aProcessed* - if `True` command is expected that has been processed and does not appear in OnRxCommand (p.12) event.

LastNakCmd (TBMG)

```
public
  LastNakCmd: Byte;
```

The last command returned in NAK.

LastNakError (TBMG)

```
public
  LastNakError: Word;
```

The last error returned in NAK.

TBMGClient

```
type
  TBMGClient = class( TBMG (p.11) );
```

The object solves the client connection to the BMG server and using its functions.

Unit CommConnect

Unit defines classes enabling connecting to a hardware device (serial port, modem, etc.)

TCommHandle

```
type
  TCommHandle = class( TCommunicationConnection );
```

Object uses hCommDev (p.14) handle and implements all necessary functions for accessing of ports and devices opened using any function returning the handle (for ex. Windows.FileOpen or libc.open). There are supported both synchronous and asynchronous operations (multi-threaded). Many parameters can be set (baud rate, number of stop bits, parity, timeouts, events, etc.).

Baudrate (TCommHandle)

```
published
  property Baudrate: TBaudrate (p.19) read write
      default br9600;
```

Baud rate to be used.

Parity (TCommHandle)

```
published
  property Parity: TParity (p.19) read write
      default paNone;
```

Parity checking to be used.

Stopbits (TCommHandle)

```
published
  property Stopbits: TStopbits (p.19) read write
      default sb10;
```

Number of stop bits

Databits (TCommHandle)

```
published
  property Databits: TDatabits (p.20) read write
      default da8;
```

Number of databits

Options (TCommHandle)

```
published
  property Options: TCommOptions (p.Chyba! Záložka není definována.) read write;
```

Event options

DontSynchronize (TCommHandle)

```
published
  property DontSynchronize;
```

If the value is `True` all object the events are called in `TCommEventThread`, if the value is `False` the events re called by means of the `TThread.Synchronize` method in the VCL thread. The VCL thread can use visual components.

hCommDev (TCommHandle)

```
public
  property hCommDev: THandle read write;
```

Assigned device handle.

ComError2 (TCommHandle)

```
public
  procedure ComError2( const aFunc: string );
```

Raises `EComError` (p.18)

OutQueCount (TCommHandle)

```
public
  function OutQueCount{}: Integer;
```

Number of chars in output queue

Lock (TCommHandle)

```
public
  procedure Lock;
```

Starts critical section. Other thread cannot interrupt this section

See also
`Unlock` (p.14)

Unlock (TCommHandle)

```
public
  procedure Unlock;
```

Terminates the critical section.

See also
`Lock` (p.14)

OnBreak (TCommHandle)

```
published
  property OnBreak: TNotifyEvent read write;
```

Event called from the `HandleCommEvent` when condition detected.

See also
`DontSynchronize` (p.14) `TCommEventType`

OnCts (TCommHandle)

```
published
  property OnCts: TNotifyEvent read write;
```

Event called from the `HandleCommEvent` when condition detected.

See also
`DontSynchronize` (p.14) `TCommEventType`

OnDsr (TCommHandle)

```
published
  property OnDsr: TNotifyEvent read write;
```

Event called from the `HandleCommEvent` when condition detected.

See also
`DontSynchronize` (p.14) `TCommEventType`

OnRing (TCommHandle)

```
published
  property OnRing: TNotifyEvent read write;
```

Event called from the `HandleCommEvent` when condition detected.

See also
`DontSynchronize` (p.14) `TCommEventType`

OnRlsd (TCommHandle)

```
published
  property OnRlsd: TNotifyEvent read write;
```

Event called from the `HandleCommEvent` when condition detected.

See also
`DontSynchronize` (p.14) `TCommEventType`

OnError (TCommHandle)

```
published
  property OnError: TCommErrorEvent (p.Chyba! Záložka není definována.) read write;
```

Event called from the `HandleCommEvent` when condition detected.

See also
`DontSynchronize` (p.14) `TCommEventType`

OnRxChar (TCommHandle)

```
published
  property OnRxChar;
```

Event called from the `HandleCommEvent` when a char received and is in input queue.

See also
`DontSynchronize` (p.14) `TCommEventType` `Retrieve` `InQueCount`

OnRxFlag (TCommHandle)

```
published
  property OnRxFlag: TNotifyEvent read write;
```

Event called from the `HandleCommEvent` when condition detected.

See also
`DontSynchronize` (p.14) `TCommEventType`

OnTxEmpty (TCommHandle)

```
published
  property OnTxEmpty: TNotifyEvent read write;
```

Event called from the `HandleCommEvent` when last char from output queue has been sent

See also
`DontSynchronize` (p.14) `TCommEventType` `Send` `OutQueCount` (p.14)

TComm

```
type
  TComm = class( TCommHandle (p.13) );
```

Object implements the device opening and closing by means of WIN32 API `Windows.OpenFile` function or Linux `libc.open`.

DeviceName (TComm)

```
published
  property DeviceName: string read write;
```

Name of device for the `Windows.OpenFile` API function or `libc.open` in Linux

TModem

```
type
  TModem = class( TComm (p.16) );
```

Object for dial-up connection to dial-up host server. In `OpenConn` method is dialed `PhoneNumber` (p.18) and is waiting for modem `CONNECT` response. After that is set `Active` to `True`.

cInit (TModem)

```
published
  property cInit: TString read write;
```

Initialization string sent to modem during initialization, default `ATZ`.

DelayBeforeInit (TModem)

```
published
  property DelayBeforeInit: Integer read write;
```

Delay in *ms* before initializing.

DelayAfterInit (TModem)

```
published
  property DelayAfterInit: Integer read write;
```

Delay in *ms* after initializing.

ResponseTimeout (TModem)

```
published
  property ResponseTimeout: Integer read write;
```

Timeout in *ms* in that modem must response to command

ConnectTimeout (TModem)

```
published
  property ConnectTimeout: Integer read write;
```

Timeout in *ms* in that modem must make success connection (`rConnect` (p.17) string is expected).

rInit (TModem)

```
published
  property rInit: TString read write;
```

Response to initialization command, default `OK`.

cDial (TModem)

```
published
  property cDial: TString read write;
```

Initialization string sent to modem when dialing, default `ATM1L1X3DT` (speaker on when dialing, no dial tone, tone dialing), `PhoneNumber` (p.18) follows.

rConnect (TModem)

```
published
  property rConnect: TString read write;
```

Response when successfully connected, default `CONNECT` (connection is done when received string `CONNECT`, `CONNECT 9600`, `CONNECT MNP10`, etc.)

rBusy (TModem)

```
published
  property rBusy: TString read write;
```

Response if line is busy, default `BUSY`.

rNoCarrier (TModem)

```
published
  property rNoCarrier: TString read write;
```

No carrier response, default `NO CARRIER`.

rNoDialtone (TModem)

```
published
  property rNoDialtone: TString read write;
```

No dial tone response, default `NO DIALTONE`. See also `cDial` (p.17) and `ATX3` command.

cHangUp (TModem)

```
published
  property cHangUp: TString read write;
```

Hangup command, default `+++ATH`.

rHangUp (TModem)

```
published
  property rHangUp: TString read write;
```

Hangup response, default `OK`.

PhoneNumber (TModem)

```
published
  property PhoneNumber: TString read write;
```

Phone number to be dialed. Number is attached to `cDial` (p.17). Comma means pause in dialing, for ex. `0,02123456`.

IsMakingCall (TModem)

```
public
  property IsMakingCall: Boolean read;
```

If `True` modem is just making connection (dialing).

Drop (TModem)

```
public
  procedure Drop;
```

Drops connection, if dialing drops it (in this moment `Active` is `False`)

OnRxCommand (TModem)

```
published
  property OnRxCommand: TModemRxCommandEvent (p.Chyba! Záložka není definována.) read write;
```

Event is called when modem is in command mode (when `Active` is `False`). When connection is active, `OnRxChar` (p.15) event is called. Event is processed in COM thread.

EComError

```
type
  EComError = class( EConnectError );
```

Exception raised when communication error occurs

TBaudrate

```
type
  TBaudrate =
    ( br110
    , br300
    , br600
    , br1200
    , br2400
    , br4800
    , br9600
    , br14400
    , br19200
    , br38400
    , br56000
    , br57600
    , br115200
    , br128000
    , br256000
    );
```

Enumerates possible baud rates. Note that primary rates are Microsoft Windows rates hence not all Linux (termios) rates are supported.

TParity

```
type
  TParity =
    ( paNone
    , paOdd
    , paEven
    , paMark
    , paSpace
    );
```

Enumaretes possible parity options. Note that primary is Microsoft Windows OS hence not all options are supported in Linux (termios).

TStopbits

```
type
  TStopbits =
    ( sb10
    , sb15
    , sb20
    );
```

Enumerates possible stop bit options. Note that primary is Microsoft Windows OS hence not all options are supported in Linux (termios).

TDatabits

```
type
  TDatabits =
    ( da4
    , da5
    , da6
    , da7
    , da8
    );
```

Enumerates possible data bit options. Note that primary is Microsoft Windows OS hence not all options are supported in Linux (termios).

TFlowControl

```
type
  TFlowControl =
    ( fcNone
    , fcCTS
    , fcDTR
    , fcSoftware
    , fcDefault
    );
```

Enumerates possible flow control options. Note that primary is Microsoft Windows OS hence not all options are supported in Linux (termios).

Unit EMI

The SMSC *External Machine Interface* (EMI) is based on an extended subset of the UCP protocol defined for the ERMES paging system in *ETS 300 133 3*. When referring to 'UCP' in the context of the SMSC, almost always the EMI, the extended subset of the ERMES UCP, is meant.
In order to provide access to the more extensive set of SMS commands, it was necessary to extend the UCP definition with some additional, SMSC specific commands, such as 'SMS message transfer operation' and 'SMT alert operation'
There is implemented support of command operations 31, 51, 52, 53, 60. Underlying protocol is TCP/IP protocol and X.25.
EMI connection was tested for *T-Online* and *Oskar* operators.

TEMIClient

```
type
  TEMIClient = class( TLogConnection );
```

Object for client connection to SMS center (SMSC).

Params (TEMIClient)

```
published
  property Params: TStrings read write;
```

Parameters necessary for connecting, see EMI documentation and SMSC configuration

User=<user account>

Short number identifying account. If the parameter value is empty is expected anonymous login. Only keep-alive test is done to test connection on login.

Password=<password>

Account password

SMSC_ADC=<AdC>

Default: *User*
Address (phone number) assigned to service (OA/DA number)

SMT_KEEPALIVE_INTERVAL=<sec>

Default: `300 sec`
In this interval is periodically tested connection to SMSC. Client (SMT) sends keep-alive messages, see *SMT_KEEPALIVE_CMD*

SMT_KEEPALIVE_CMD=<cmd>

Default: `31`
Client's (SMT) keep-alive operation (SMT->SMSC).

`31` use Alert operation (for ex. T-Mobile)
`51` use Submit Short Message operation (for ex. Oskar)

SMT_KEEPALIVE_DA=<number>
Destination number/address of keep-alive SM. It has meaning only if *SMT_KEEPALIVE_CMD* is `51`

SMT_KEEPALIVE_OA=<number>
Originated number/address of keep-alive SM. It has meaning only if *SMT_KEEPALIVE_CMD* is `51`

SMT_KEEPALIVE_TEXT=<text>
Text of keep-alive SM. It has meaning only if *SMT_KEEPALIVE_CMD* is `51`

SMSC_KEEPALIVE_CMD=<cmd>
Default: `-1`
Server's (SMSC) keep-alive operation (SMSC->SMT).
`52` listen specific Delivery Short Message operation (for ex. Oskar)

SMSC_KEEPALIVE_OA=<number>
Originated number/address of keep-alive SM. It has meaning only if *SMSC_KEEPALIVE_CMD* is `52`

SMSC_KEEPALIVE_DA=<number>
Destination number/address of keep-alive SM. It has meaning only if *SMSC_KEEPALIVE_CMD* is `52`

SMSC_KEEPALIVE_TEXT=<text>
Text of keep-alive SM. It has meaning only if *SMSC_KEEPALIVE_CMD* is `52`

SMT_INT_PREFIX=<prefix>
International sign (plus, +) will be replaced by this prefix.
For example if parameter is `00`, number `+420602123456` will be changed to `00420602123456`. If parameter is empty sign *plus* is removed. Depends on EMI implementation.

ComDevice (TEMIClient)

```
published
  property ComDevice: TCommunicationConnection read write;
```

The device that is used for connecting to the EMI (for ex. `TClientSocketConnection` (p.53))

RepeatCount (TEMIClient)

```
published
  property RepeatCount: Integer read write
      default 5;
```

Number of attempts to send command.

See also
RepeatTimeout

RecTimeout (TEMIClient)

```
published
  property RecTimeout: LongInt read write
      default 3000;
```

When a command is send, TEMIClient is waiting for response max. *RecTimeout*.

ETXTimeout (TEMIClient)

```
published
  property ETXTimeout: LongInt read write
      default 1000;
```

Auxiliary timeout used in `SendCommand` and some command is received (waiting for `ETX`).

RepeatTimeout (TEMIClient)

```
published
  property RepeatTimeout: LongInt read write
      default 500;
```

Timeout used when command is repeated

See also

`RepeatCount` (p.22)

IsBusy (TEMIClient)

```
public
  property IsBusy: Boolean read;
```

Check if component is currently busy

See also

`Busy` (p.23), `Unbusy` (p.24)

PhoneNumber (TEMIClient)

```
public
  property PhoneNumber: string read;
```

Client phone number / alias, for ex. `5071`). It's value of the *SMSC_ADC* `Params` (p.21).

CommandBufferCount (TEMIClient)

```
public
  property CommandBufferCount: Integer read;
```

Number of received commands in buffer

Busy (TEMIClient)

```
public
  procedure Busy;
```

Lock component's busy flag

See also

`Unbusy` (p.24), `IsBusy` (p.23)

Unbusy (TEMIClient)

```
public
  procedure Unbusy;
```

Unlock component's busy flag

See also

Busy (p.23), IsBusy (p.23)

smTestConnection (TEMIClient)

```
public
  function smTestConnection{}: Boolean;
```

Returns conection status, see. LastSendTick (p.25).

smSubmit (TEMIClient)

```
public
  function smSubmit(
      const aPhn: string;
      const aOAPhn: string;
      const Msg: string;
      aBinary: Boolean;
      aMCLs: Byte;
      aPID: Byte;
      aNT: Byte;
      aRPI: Byte;
      Validity: TDateTime ): TDateTime;
```

Sends SMS *Msg* to *aPhn*. Next parameters see. *EMI specification*. Returns SCTS time stamp, that was assigned to message in SMS center. There is possible to require message delivery notification using *aNT* parameter. Notification is delivered as special message. For *aMCLs* use `mclxxxx` constants.

`aPhn`:
: Target phone number

`aOAPhn`:
: Originator phone number, if empty default PhoneNumber (p.23) is used

`Msg`:
: Message

`aNT`:
: see *ntXXX constants*

OnBusyChanged (TEMIClient)

```
published
  property OnBusyChanged: TBusyChanged (p.Chyba! Záložka není definována.) read write;
```

Notification when busy flag is changed

See also

Busy (p.23), Unbusy (p.24), IsBusy (p.23)

OnRxChar (TEMIClient)

```
published
  property OnRxChar: TRxCharEvent (p.Chyba! Záložka není definována.) read write;
```

Is called in the VCL thread when a char was received.

OnRxCommand (TEMIClient)

```
published
  property OnRxCommand: TRxCommandEvent (p.Chyba! Záložka není definována.) read write;
```

Is called in the VCL thread when a command was received. Params *aOT*-command number (`cmdsDelivery` or `cmdsDeliveryNotification`), *aTRN* - transaction number (now always 0), *aResult* - response / query, *aData*. See also `OnRxCommandBeforeAcknowledge` (p.25). Using `GetDeliverySMItem` or `GetDeliveryNotificationItem` functions is possible decode SMS content.

OnRxCommandBeforeAcknowledge (TEMIClient)

```
published
  property OnRxCommandBeforeAcknowledge: TRxCommandBeforeAcknowledgeEvent (p.Chyba! Záložka není
```

Is called in the COM thread when a command was received. Params *aOT*-command number (`cmdsDelivery` or `cmdsDeliveryNotification`), *aTRN* - transaction number (now always 0), *aResult* - response / query, *aData*, *aProcessed* - if `True` command is expected that has been processed and does not appear in `OnRxCommand` (p.25) event. Using `GetDeliverySMItem` or `GetDeliveryNotificationItem` functions is possible decode SMS content.

LastNakCmd (TEMIClient)

```
public
  LastNakCmd: Byte;
```

The last command returned in NAK.

LastNakError (TEMIClient)

```
public
  LastNakError: Integer;
```

The last error returned in NAK.

LastSendTick (TEMIClient)

```
public
  LastSendTick: DWord;
```

Windows `GetTickCount` value when was received last char. EMI server disconnects when no char received from client in `EMIDisconnectTimeout`. Client can periodically test connection using `smTestConnection` (p.24) not to disconnect socket.

mclDisplay

```
const
  mclDisplay = 0;
```

Message class 0

mclME

```
const
  mclME = 1;
```

Message class 1 (memory equipment)

mclSIM

```
const
  mclSIM = 2;
```

Message class 2 (store in SIM)

mclTE

```
const
  mclTE = 3;
```

Message class 3 (terminal equipment)

mclDefault

```
const
  mclDefault = 'FF';
```

default message class, safe option for all mobiles (some mobiles do not display non default value)

Unit EMIX

EMIX.OCX is ActiveX library developed in Delphi and can be used in any application that supports ActiveX components (Visual Basic, MS Word, etc.).
Before using it's necessary to register ActiveX using common Windows tool `regsvr32 emix.ocx`.
In non-Delphi environment is necessary to register also *stdvcl32.dll* library.

TEMIClientX

```
type
  TEMIClientX = class( TOleComponent );
```

TEMIClientX is a non-visual component for connecting to SMSC center using EMI protocol (TCP/IP). Component support both apartment and free threading model. The component provides functionality of `TEMIClient` (p.21) and `TClientSocketConnection` (p.53).

Params (TEMIClientX)

```
published
  property Params: IStrings read write;
```

In *Params* are stored all EMI protocol configuration parameters. It's possible setup all `TEMIClient` (p.21).`Params` (p.21) parameters and in addition following ones:

IP=<ip>
IP address of SMSC, see also `TClientSocketConnection` (p.53).`Address` (p.53)

PORT=<port>
Port of SMSC, see also `TClientSocketConnection` (p.53).`Port` (p.53)

RECONNECT_TIMEOUT=<sec>
Default: `60`
Interval that is used when connection has been broken and component does periodical attempts for reconnection. If `0` no reconnecting.

SMT_KEEPALIVE_INTERVAL=<sec>
Default: `300`
The component is sending keep-alive datagrams to inform SMSC that connection is alive. If `0` keep-alive notification is disabled.

ETX_TIMEOUT=<msec>
Default: `1000`
See `TEMIClient` (p.21).`ETXTimeout` (p.23)

REC_TIMEOUT=<msec>
Default: `3000`
See `TEMIClient` (p.21).`RecTimeout` (p.23)

REPEAT_COUNT=<times>
Default: `2`
See `TEMIClient` (p.21).`RepeatCount` (p.22)

REPEAT_TIMEOUT=<msec>
Default: `500`
See `TEMIClient` (p.21).`RepeatTimeout` (p.23)

Active (TEMIClientX)

```
public
  property Active: WordBool read write;
```

See `TEMIClient` (p.21).Active

LastErrorStr (TEMIClientX)

```
public
  property LastErrorStr: WideString read;
```

Reason of last error

See also
`Submit` (p.28)

Busy (TEMIClientX)

```
public
  procedure Busy;
```

See `TEMIClient` (p.21).`Busy` (p.23)

Unbusy (TEMIClientX)

```
public
  procedure Unbusy;
```

See `TEMIClient` (p.21).`Unbusy` (p.24)

IsBusy (TEMIClientX)

```
public
  function IsBusy{}: WordBool;
```

See `TEMIClient` (p.21).`IsBusy` (p.23)

Submit (TEMIClientX)

```
public
  function Submit(
      const TargetNumber: WideString;
      const Msg: WideString;
      Binary: WordBool;
      MCLs: Byte;
      PID: Byte;
      NT: Byte;
      RPI: Byte;
      Validity: TDateTime ): TDateTime;
```

See `TEMIClient` (p.21).`smSubmit` (p.24)

If an error occurs result is *0*. Check `LastErrorStr` (p.28) property to obtain error message.

TestConnection (TEMIClientX)

```
public
  function TestConnection{}: WordBool;
```

See `TEMIClient` (p.21).`smTestConnection` (p.24)

SetParam (TEMIClientX)

```
public
  procedure SetParam(
      const Name: WideString;
      const Value: WideString );
```

Set one parameter to `Params` (p.27).

GetParam (TEMIClientX)

```
public
  function GetParam( const Name: WideString ): WideString;
```

Read one parameter from `Params` (p.27)

OnBusyChanged (TEMIClientX)

```
published
  property OnBusyChanged: TEMIClientXOnBusyChanged (p.Chyba! Záložka není definována.) read write;
```

See `TEMIClient` (p.21).`OnBusyChanged` (p.24)

OnLog (TEMIClientX)

```
published
  property OnLog: TEMIClientXOnLog (p.Chyba! Záložka není definována.) read write;
```

Enables logging of communication into file or a terminal. Use a `TLogger` object because event is fired in not VCL thread.

See also

`TLogger.PreformatText`

OnRxCommand (TEMIClientX)

```
published
  property OnRxCommand: TEMIClientXOnRxCommand (p.Chyba! Záložka není definována.) read write;
```

See `TEMIClient` (p.21).`OnRxCommand` (p.25)

There is one more parameter *Item* that contains SMS message (if non equal to `Null`).

Note: *Item* is passed by reference due to functionality only, no effect if value changed in event

See also

`IEMIRetrieveDeliveryItem` (p.30), `IEMIRetrieveDeliveryNotificationItem` (p.30)

OnRxCommandBeforeAcknowledge (TEMIClientX)

```
published
  property OnRxCommandBeforeAcknowledge: TEMIClientXOnRxCommandBeforeAcknowledge (p.Chyba! Zál
```

See `TEMIClient` (p.21).`OnRxCommandBeforeAcknowledge` (p.25)

There is one more parameter *Item* that contains SMS message (if non equal to `Null`).
Note: *Item* is passed by reference due to functionality only, no effect if value changed in event `Processed` parameter

`emipfUnprocessed` command/SM is stored in command buffer for `OnRxCommand` (p.29) event
`emipfProcessed` command/SM is not stored in command buffer for `OnRxCommand` (p.29) event
`emipfAborted` command/SM is not acknowledged to SMCS (SMSC will send again command/SM in future)

See also

`IEMIRetrieveDeliveryItem` (p.30), `IEMIRetrieveDeliveryNotificationItem` (p.30)

IEMIRetrieveDeliveryItem

```
type
  IEMIRetrieveDeliveryItem = interface( IDispatch );
```

Data of received SM, See *EMI documentation*

See also

`TEMIClient` (p.21).`OnRxCommand` (p.25), `TEMIClient` (p.21).`OnRxCommandBeforeAcknowledge` (p.25), `IEMIRetrieveDeliveryNotificationItem` (p.30)

IEMIRetrieveDeliveryNotificationItem

```
type
  IEMIRetrieveDeliveryNotificationItem = interface( IDispatch );
```

Data of received SM notification. See *EMI documentation*

See also

`TEMIClient` (p.21).`OnRxCommand` (p.25), `TEMIClient` (p.21).`OnRxCommandBeforeAcknowledge` (p.25), `IEMIRetrieveDeliveryItem` (p.30)

Unit GDEP

The SMSC *Game Data Exchange Protocol* (GDEP) is based on a SMPP protocol (currently version 1.5).
GDEP support sending and receiving of SM, MMS, processing WAP, managing game accounts.
There is implemented support for plain HTTP (via VPN) or for HTTPS (via public internet).
GDEP connection was tested for *Orange Slovakia* operator.

TGDEPClient

```
type
  TGDEPClient = class( TLogConnection );
```

Object for client connection to SMS center (SMSC).

Unit GSM

The GSM objects facilitate a communication with GSM modems. Using these objects the modem logs in a GSM network, can be controlled via AT commands, SMS can be sent and received.

TSMS

```
type
  TSMS = class;
```

Abstract object that implements the SMS message features (both the SMS and cell broadcasts)
Parameter and constant purpose is described in *GSM and SMS documentation*.

Get2Bits (TSMS)

```
protected
  function Get2Bits(
      aVar: Byte;
      Index: Integer ): Byte;
```

Returns one bit from *aVars*.

Set2Bits (TSMS)

```
protected
  procedure Set2Bits(
      var aVar: Byte;
      Index: Integer;
      Value: Byte );
```

Returns double bits from *aVars*

EncodePDU (TSMS)

```
protected
  function EncodePDU{}: TString; virtual; abstract;
```

Encodes PDU message according to object values.

DecodePDU (TSMS)

```
protected
  procedure DecodePDU( const Value: TString ); virtual; abstract;
```

Decodes PDU string a stores values to object.

TSMS2

```
type
  TSMS2 = class( TSMS (p.32) );
```

Class adds features for all non-cell broadcasts messages.
Phone numbers in OA or DA properties can be international - prefixed `'+'`, local (national) - no prefix or alias - prefixed `'#'`.

Examples:
0602123456 national +420602123456 international #4616 alias, short number

CreateSMS (TSMS2)

```
public
  class function CreateSMS(
      aOwner: TGSM (p.34);
      const aPDU: TString;
      aMTDir: Boolean ): TSMS2 (p.32);
```

Creates correct TSMS2 (p.32) class instance according to type of PDU message

`aMTDir`:

If `True` expected message direction is *mobile terminated* SMS.

TSMSDeliver

```
type
  TSMSDeliver = class( TSMS2 (p.32) );
```

Implements the mobile terminated (SMS-DELIVER, received) SMS features.

TSMSSubmit

```
type
  TSMSSubmit = class( TSMS2 (p.32) );
```

Implements the mobile originating (SMS-SUBMIT) SMS features.

TSMSStatusReport

```
type
  TSMSStatusReport = class( TSMS2 (p.32) );
```

Implements the SMS-STATUS-REPORT features.

TSMSCommand

```
type
  TSMSCommand = class( TSMS2 (p.32) );
```

Implements the SMS-COMMAND features.

TSMSDeliverReport

```
type
  TSMSDeliverReport = class( TSMS2 (p.32) );
```

Implements the SMS-DELIVERY-REPORT features.

TSMSSubmitReport

```
type
  TSMSSubmitReport = class( TSMSDeliverReport (p.33) );
```

Implements the SMS-SUBMIT-REPORT features.

TCellBroadcast

```
type
  TCellBroadcast = class( TSMS (p.32) );
```

Implements the cell broadcast features.

TGSMAsyncThread

```
type
  TGSMAsyncThread = class( TThread );
```

Thread receiving characters from TGSM (p.34).COMDevice (p.35).

TGSM

```
type
  TGSM = class( TConnection );
```

Object communicates with a GSM modem. Now are supported these modules (modems):

- Siemens A1
- Siemens M1
- Siemens M20
- Siemens TC35
- Ericsson GM12
- Nokia 6210
- Nokia 9110
- Nokia 6650
- Nokia 6600 (seems no delivery status and uindOnlyIndication supported)
- Siemens M35, S35, C45, ME/S45, S55
- Motorola Timeport T260
- Wavecom
- Wavecom Fasttrack
- Fargo Maestro

The both SMS message formats are implemented, the text format and the PDU format. When the modem supports both formats, the PDU is recommended. Other modems have not been yet tested, but probably should work.
Siemens C45, ME/S45 works as *Siemens M35*. It is useful to reset the Baud rate for *ME/S45*, as they handle higher speeds, which can be customized in the mobile.

Equipment (TGSM)

```
published
  property Equipment: TGSMEquipment (p.Chyba! Záložka není definována.) read write;
```

The modem type, see *eqxxxx constants*. Set this value before SetDefaults (p.37) or Open method is called.

SMSFormat (TGSM)

```
published
  property SMSFormat: Byte read write;
```

The type of SMS format (use smsfPDU, smsfText constants)

UnsolicitedIndication (TGSM)

```
published
  property UnsolicitedIndication: TGSMUnsolicitedIndicationSet (p.Chyba! Záložka není definována.) re
```

Enabled/disables the unsolicited error notification from the modem to the computer, see AT+CNMI.

PIN (TGSM)

```
published
  property PIN: TString read write;
```

The PIN of the SIM card in the modem, see AT+CPIN.

SCA (TGSM)

```
published
  property SCA: TString read write;
```

The SMS center number (format is for ex. *+420602123456*), see AT+CSCA

COMDevice (TGSM)

```
published
  property COMDevice: TComm (p.16) read write;
```

The device that is connected modem to.

CommandEcho (TGSM)

```
published
  property CommandEcho: Boolean read write
      default True;
```

Enable/disable serial port command echo

ExtendedErrorMsg (TGSM)

```
published
  property ExtendedErrorMsg: Boolean read write
      default False;
```

see AT+CMEE command

RepeatCount (TGSM)

```
published
  property RepeatCount: Integer read write
      default 5;
```

How many times object will try to repeat sending a command if an error occurs.

RecTimeout (TGSM)

```
published
  property RecTimeout: LongInt read write
      default 6000;
```

The timeout in *ms* used when the object is waiting for character receiving.

RepeatTimeout (TGSM)

```
published
  property RepeatTimeout: LongInt read write
      default 500;
```

The timeout in *ms* used when the object is waiting for next attempt of command sending.

IsBusy (TGSM)

```
public
  property IsBusy: Boolean read;
```

Returns `True` if serial port is locked using the `Busy` (p.37) method.

See also
`Unbusy` (p.37) `OnBusyChanged` (p.41)

LastError (TGSM)

```
public
  property LastError: Integer read write;
```

Returns the last error code.

LastErrorMsg (TGSM)

```
public
  property LastErrorMsg: TString read write;
```

Returns the last error description.

ErrorCodes (TGSM)

```
public
  property ErrorCodes: TStrings read write;
```

List of possible error codes with description

NetworkRegistration (TGSM)

```
public
  property NetworkRegistration: Integer read;
```

The status code indicating current GSM network registration, see `AT+CREG`.

mem1 (TGSM)

```
public
  property mem1: TString read write;
```

Modem SMS storage location used for reading messages, see `AT+CPMS` command.

ManufacturerId (TGSM)

```
public
  property ManufacturerId: TString read GetMEId (p.Chyba! Záložka není definována.);
```

Id returned by AT+CGMI command. May be used in both active and inactive states. If is in active state value is cached to ask modem only once.

ModelId (TGSM)

```
public
  property ModelId: TString read GetMEId (p.Chyba! Záložka není definována.);
```

Id returned by AT+CGMM command. May be used in both active and inactive states. If is in active state value is cached to ask modem only once.

RevisionId (TGSM)

```
public
  property RevisionId: TString read GetMEId (p.Chyba! Záložka není definována.);
```

Id returned by AT+CGMR command. May be used in both active and inactive states. If is in active state value is cached to ask modem only once.

SerialNumberId (TGSM)

```
public
  property SerialNumberId: TString read GetMEId (p.Chyba! Záložka není definována.);
```

Id returned by AT+CGSN command. May be used in both active and inactive states. If is in active state value is cached to ask modem only once.

SetDefaults (TGSM)

```
public
  procedure SetDefaults;
```

Sets all the default modem parameters (for ex. TCommHandle (p.13).Baudrate (p.13)) according the Equipment (p.34) value.

Busy (TGSM)

```
public
  procedure Busy;
```

Locks the modem serial port. It is necessary to not to interrupt current modem communication.

See also
Unbusy (p.37) OnBusyChanged (p.41) IsBusy (p.36)

Unbusy (TGSM)

```
public
  procedure Unbusy;
```

Unlocks the modem serial port. It is necessary to not to interrupt current modem communication.

See also
Busy (p.37) OnBusyChanged (p.41) IsBusy (p.36)

SendATCommand (TGSM)

```
public
  function SendATCommand(
      const aCmd: TString;
      aWaitFor: Byte;
      aResultS: TStrings;
      RepCount: Integer = 0 ): Integer;
```

Sends the *aCmd* command to the modem and according to the *aWaitFor* argument is waiting for the modem result.

Examples:

```
  CheckAT(SendATCommand('AT+CPIN?', atrCode+atrParams+atrATResponse, Sg));
  CheckAT(SendATCommand('AT+CPIN="1243"', atrCode, nil));
  CheckAT(SendATCommand('AT+CREG?', atrCode+atrParams+atrATResponse, nil));
  CheckAT(SendATCommand('AT+CREG=1', atrCode, nil));
  CheckAT(SendATCommand('AT+CSCA="+420602909909"', atrCode, nil));
  CheckAT(SendATCommand('AT+CMGS="+420602123456">Hello', atrCode+atrParams+atrBothNecessar
  CheckAT(SendATCommand('AT+CMGR=18' atrCode+atrParams, Sg));
  CheckAT(SendATCommand('AT+CMGD=25', atrCode, nil));
  CheckAT(SendATCommand('AT+CMGL=1', atrCode+atrParams, Sg));
```

This function is not necessary call directly when sending or receiving SMS messages. Use the `SendSMS` (p.38), `ReadSMS` (p.39), `DeleteSMS` (p.39) and `GetSMSList` (p.39) functions.

`aCmd`:

AT command to be sent

`aWaitFor`:

The aWaitFor constants: `atrCode` - command returns OK or ERROR response `atrParams` - command returns some parameters `atrBothNecessary` - command returns both some parameters and the `OK/ERROR` response `atrPDU` - command is a PDU command and modem will return prompt `' >'` `atrATResponse` - command returns parameters prefixed by `'+cmd:'` `atrCRLF` - internal meaning

`aResultS`:

When command returns parameters they are returned in the *aResultS* argument.

`RepCount`:

How many times command can be repeated if error is returned.

`result`:

The function returns a error code that can be passed through the pipe `CheckAT` (p.38) function.

CheckAT (TGSM)

```
public
  function CheckAT( aRes: Integer ): Integer;
```

The pipe function checks the *aRes* error code and the exception is fired if not `OK`.

SendSMS (TGSM)

```
public
  procedure SendSMS( aSMS: TSMS2 (p.32) );
```

Sends a SMS message, see `AT+CMGS`.

ReadSMS (TGSM)

```
public
  function ReadSMS(
      aIndex: Integer;
      var Stat: Integer ): TSMS (p.32);
```

Reads a SMS corresponding to the *aIndex* argument. In the *aStat* is returned a SMS status code (read, sent, unsent, unread, etc.), see AT+CMGR.

See also
mem1 (p.36)

DeleteSMS (TGSM)

```
public
  procedure DeleteSMS( aIndex: Integer );
```

Deletes a SMS message from the modem list, see AT+CMGD.

GetSMSList (TGSM)

```
public
  function GetSMSList( aStat: Integer ): TStrings;
```

Returns the list of SMS messages corresponding the *aStat* argument (read, sent, unsent, unread, etc.), see AT+CMGL.

aStat:
: kind of SMSes according its status code (read, sent, unsent, unread, etc.) to be retrieved

result:
: TStringList collection containing retrieved SMSes. Format of item is <idx>=<stat>. Where idx> is index (identifier) returned by modem. Use this number to reference SMS in ReadSMS (p.39) and DeleteSMS (p.39) or when using direct AT commands. <stat> is status of the SMS (the same as *aStat*). If SMS contents was retrieved it's stored into a TSMS (p.32) object that is assigned to appropriate *Objects[]* item.

See also
mem1 (p.36)

SwitchOff (TGSM)

```
public
  procedure SwitchOff;
```

Sends the software switch-off modem command. It is not supported by all the GSM modems.

GetSupportedSMSFormats (TGSM)

```
public
  function GetSupportedSMSFormats{}: TByteSet (p.Chyba! Záložka není definována.);
```

Returns supported SMS formats of current Equipment (p.34).

See also
smsfPDU smsfText

OpenComDevice (TGSM)

```
protected
  procedure OpenComDevice;
```

Opens assigned COMDevice (p.35) and save its setting (TCommHandle (p.13).OnRxChar (p.15), etc.).

See also

CloseComDevice

CloseComDevice (TGSM)

```
protected
  procedure CloseComDevice( aRestore: Boolean );
```

Clese assigned com device. If *aRestore* restores COMDevice (p.35) properties saved in OpenComDevice (p.40).

ProcessRxLine (TGSM)

```
protected
  procedure ProcessRxLine( const aLine: TString ); virtual;
```

Process line received from COMDevice (p.35). Procedure try to recognize unsolicited messages and error codes.

GetCMGLength (TGSM)

```
protected
  function GetCMGLength( aSMS: TSMS2 (p.32) ): Integer; virtual;
```

Returns *<length>* of AT+CMGS and AT+CMGW commands. This length is very dependant on Equipment (p.34) and firmware version. Override if errors are returned when sending messages.

SCAinPDU (TGSM)

```
protected
  function SCAinPDU{}: Boolean;
```

True if the SMS center number is required before the PDU encoded data. It may depend on the modem type and even on firmware number (for ex. *Siemens A1*).

OnRxChar (TGSM)

```
published
  property OnRxChar: TRxCharEvent (p.Chyba! Záložka není definována.) read write;
```

The event is called in the VCL thread when a char is received from the modem.

OnUnsolicitedLine (TGSM)

```
published
  property OnUnsolicitedLine: TUnsolicitedLineEvent (p.Chyba! Záložka není definována.) read write;
```

The event is called in the VCL thread when an unsolicited notification is received from the modem.

OnUnsolicitedSMS (TGSM)

```
published
  property OnUnsolicitedSMS: TUnsolicitedSMSEvent (p.Chyba! Záložka není definována.) read write;
```

The event is called in the VCL thread when a SMS is received

OnNetworkRegistration (TGSM)

```
published
  property OnNetworkRegistration: TNetworkRegistrationEvent (p.Chyba! Záložka není definována.) read
```

The event is called in the VCL thread when the GSM network registration is changed (login, logout, etc.).

See also
NetworkRegistration (p.36)

OnBusyChanged (TGSM)

```
published
  property OnBusyChanged: TBusyChanged (p.Chyba! Záložka není definována.) read write;
```

The event is called when the IsBusy (p.36) flag is changed.

See also
Busy (p.37) Unbusy (p.37)

EGSMError

```
type
  EGSMError = class( EComError (p.18) );
```

Exception raised when error occurs in TGSM (p.34).

DTToValPer

```
function DTToValPer( DT: TDateTime ): Byte;
```

Translates between the TDateTime and the SMS validity period

See also
ValPerToDT (p.41)

ValPerToDT

```
function ValPerToDT( VP: Byte ): TDateTime;
```

Translates between the TDateTime and the SMS validity period

See also
DTToValPer (p.41)

StripATResponse

```
function StripATResponse( const S: TString ): TString;
```

Strips modem AT response prefix.

See also
IsATResponse (p.42)

IsATResponse

```
function IsATResponse( const S: TString ): Boolean;
```
Returns `True` if string *S* is modem response, for ex. `'+CNMI:'` or `'CMSO:'`.

See also
StripATResponse (p.41)

ExtractParam

```
function ExtractParam(
    const S: TString;
    var Pos: Integer ): TString;
```
Extract integer parameter set from string *S* received from modem, for ex. `+CNMI: 2,1,0,0,0`.
Equivavelent of `ExtractFieldName` but delimiter is comma (',').

ExtractParamSet

```
function ExtractParamSet(
    const S: TString;
    var Pos: Integer ): TByteSet (p.Chyba! Záložka není definována.);
```
Extract interger parameter set from string *S* received from modem, for ex. `+CNMI: (0-3),(0-3),(0-2),0,(0-1)`.

See also
ExtractParam (p.42)

TrimQuotes

```
function TrimQuotes( const S: TString ): TString;
```
Trim quotes chars (`"`)

GSMError

```
procedure GSMError( const Msg: string );
```
Raises EGSMError

ISOtoSMS

```
function ISOtoSMS( ISOstring: TString ): TString;
```
Translates *ISOString* from ISO to GSM code page.

See also
SMSToISO

SMStoISO

`function SMStoISO( SMSstring: TString ): TString;`
Translates *SMSString* from GSM to ISO code page.

See also
ISOToSMS

Str2GMSEquipment

`function Str2GMSEquipment( const S: TString ): TGSMEquipment` (p.Chyba! Záložka není definována.)`;`
Converts short equipment name to equipment enumeration.

See also
GSMEquipmentShortName (p.43)

StatS

```
const
  StatS: array[ 0 .. 4 ] of TString=
    ( 'REC UNREAD'
    , 'REC READ'
    , 'STO UNSENT'
    , 'STO SENT'
    , 'ALL'
    );
```
Enumeration of strings used in <*stat*> parameter in `AT+CMGL` and `AT+CMGR`.

GSMEquipmentShortName

```
const
  GSMEquipmentShortName: array[ TGSMEquipment (p.Chyba! Záložka není definována.) ] of string=
    ( 'A1'
    , 'M1'
    , 'M20'
    , 'GM12'
    , 'Nokia6210'
    , 'Nokia6650'
    , 'Nokia9110'
    , 'M35'
    , 'S25'
    , 'T260'
    , 'WaveCom'
    , 'TC35'
    , 'Fasttrack'
    );
```

List of equipment short names, usable for ex. for storing to INI files

GSMEquipmentLongName

```
const
  GSMEquipmentLongName: array[ TGSMEquipment (p.Chyba! Záložka není definována.) ] of string=
    ( 'Siemens A1'
    , 'Siemens M1'
    , 'Siemens M20'
    , 'Ericsson GM12'
    , 'Nokia 6210'
    , 'Nokia 6650'
    , 'Nokia 9110'
    , 'Siemens M35'
    , 'Siemens S25'
    , 'Motorola T260'
    , 'WaveCom'
    , 'Siemens TC35'
    , 'Wavecom Fasttrack'
    );
```

List of equipment long names, usable for ex. for combo boxes

Unit GSMX

GSMX.OCX is ActiveX library developed in Delphi and can be used in any application that supports ActiveX components (Visual Basic, MS Word, etc.).
Before using it's necessary to register ActiveX using common Windows tool `regsvr32 gsmx.ocx`.
In non-Delphi environment is necessary to register also `stdvcl32.dll` library.
The library uses the same name conventions as GSM (p.32) unit. There are only minor differences:

- set of bytes -> bit mask (in word)
- strings -> WideString
- Boolean -> WordBool
- classes -> interfaces

TGSMX

```
type
  TGSMX = class( TOleComponent );
```

TEMIClientX is a non-visual component for connecting to GSM module. Component support both apartment and free threading model. The component provides functionality of TGSM (p.34) and TComm (p.16).

Active (TGSMX)

```
published
  property Active: WordBool read write stored False;
```

See TGSM (p.34).Active

ErrorCodes (TGSMX)

```
published
  property ErrorCodes: IStrings read write stored False;
```

See TGSM (p.34).ErrorCodes (p.36)

Mem1 (TGSMX)

```
published
  property Mem1: WideString read write stored False;
```

See TGSM (p.34).mem1 (p.36)

Equipment (TGSMX)

```
published
  property Equipment: TOleEnum read write stored False;
```

See TGSM (p.34).Equipment (p.34)

SMSFormat (TGSMX)

```
published
  property SMSFormat: TOleEnum read write stored False;
```

See TGSM (p.34).SMSFormat (p.35)

UnsolicitedIndication (TGSMX)

```
published
  property UnsolicitedIndication: Integer read write stored False;
```

See TGSM (p.34).UnsolicitedIndication (p.35)

There is difference because ActiveX property is treated as bitmask (not set of bytes)

PIN (TGSMX)

```
published
  property PIN: WideString read write stored False;
```

See TGSM (p.34).PIN (p.35)

SCA (TGSMX)

```
published
  property SCA: WideString read write stored False;
```

See TGSM (p.34).SCA (p.35)

CommandEcho (TGSMX)

```
published
  property CommandEcho: WordBool read write stored False;
```

See TGSM (p.34).CommandEcho (p.35)

ExtendedErrorMsg (TGSMX)

```
published
  property ExtendedErrorMsg: WordBool read write stored False;
```

See TGSM (p.34).ExtendedErrorMsg (p.35)

RepeatCount (TGSMX)

```
published
  property RepeatCount: Integer read write stored False;
```

See TGSM (p.34).RepeatCount (p.35)

RecTimeout (TGSMX)

```
published
  property RecTimeout: Integer read write stored False;
```

See TGSM (p.34).RecTimeout (p.36)

RepeatTimeout (TGSMX)

```
published
  property RepeatTimeout: Integer read write stored False;
```

See TGSM (p.34).RepeatTimeout (p.36)

DeviceName (TGSMX)

```
published
  property DeviceName: WideString read write stored False;
```

See TComm (p.16).DeviceName (p.16)

Parity (TGSMX)

```
published
  property Parity: TOleEnum read write stored False;
```

See TCommHandle (p.13).Parity (p.13)

StopBits (TGSMX)

```
published
  property StopBits: TOleEnum read write stored False;
```

See TCommHandle (p.13).Stopbits (p.13)

BaudRate (TGSMX)

```
published
  property BaudRate: TOleEnum read write stored False;
```

See TCommHandle (p.13).Baudrate (p.13)

DataBits (TGSMX)

```
published
  property DataBits: TOleEnum read write stored False;
```

See TCommHandle (p.13).Databits (p.13)

CheckInterval (TGSMX)

```
published
  property CheckInterval: Integer read write stored False;
```

TGSMX (p.45) checks periodicaly the GSM module in CheckInterval (in *sec*) and rereads received SMS (GetSMSList (p.49)). Even UnsolicitedIndication (p.46) and OnUnsolicitedSMS (p.50) event are applied some SMS may leeave uncaught (and stored in module memory). If a unread SMS is obtained then is passed to OnUnsolicitedSMS (p.50) event.
If CheckInterval is zero, no checking is processed.

LastError (TGSMX)

```
public
  property LastError: Integer read;
```

See TGSM (p.34).LastError (p.36)

IsBusy (TGSMX)

```
public
  property IsBusy: WordBool read;
```

See TGSM (p.34).IsBusy (p.36)

LastErrorMsg (TGSMX)

```
public
  property LastErrorMsg: WideString read;
```

See TGSM (p.34).LastErrorMsg (p.36)

NetworkRegistration (TGSMX)

```
public
  property NetworkRegistration: Integer read;
```

See TGSM (p.34).NetworkRegistration (p.36)

ManufacturerId (TGSMX)

```
public
  property ManufacturerId: WideString read;
```

See TGSM (p.34).ManufacturerId (p.37)

ModelId (TGSMX)

```
public
  property ModelId: WideString read;
```

See TGSM (p.34).ModelId (p.37)

RevisionId (TGSMX)

```
public
  property RevisionId: WideString read;
```

See TGSM (p.34).RevisionId (p.37)

SerialNumberId (TGSMX)

```
public
  property SerialNumberId: WideString read;
```

See TGSM (p.34).SerialNumberId (p.37)

SupportedSMSFormats (TGSMX)

```
public
  property SupportedSMSFormats: Word read;
```

See TGSM (p.34).GetSupportedSMSFormats (p.39)

Busy (TGSMX)

```
public
  procedure Busy;
```

See TGSM (p.34).Busy (p.37)

Unbusy (TGSMX)

```
public
  procedure Unbusy;
```

See TGSM (p.34).Unbusy (p.37)

SetDefaults (TGSMX)

```
public
  procedure SetDefaults;
```

See TGSM (p.34).SetDefaults (p.37)

SendATCommand (TGSMX)

```
public
  function SendATCommand(
      const Cmd: WideString;
      WaitFor: Byte;
      var ResultS: IStrings;
      RepCount: Integer ): Integer;
```

See TGSM (p.34).SendATCommand (p.38)

SendSMS (TGSMX)

```
public
  procedure SendSMS( const SMS: ISMS2 );
```

See TGSM (p.34).SendSMS (p.38)

ReadSMS (TGSMX)

```
public
  function ReadSMS(
      Index: Integer;
      out Stat: Integer ): ISMS;
```

See TGSM (p.34).ReadSMS (p.39)

DeleteSMS (TGSMX)

```
public
  procedure DeleteSMS( Index: Integer );
```

See TGSM (p.34).DeleteSMS (p.39)

GetSMSList (TGSMX)

```
public
  function GetSMSList( Stat: Integer ): IStrings2 (p.50);
```

See TGSM (p.34).GetSMSList (p.39)

SwitchOff (TGSMX)

```
public
  procedure SwitchOff;
```

See TGSM (p.34).SwitchOff (p.39)

SendChar (TGSMX)

```
public
  procedure SendChar( const S: WideString );
```

Send raw characters to GSM module. If object is not Active (p.45) opens communication port (only). Module response is passed to OnRxChar (p.49) event.

OnRxChar (TGSMX)

```
published
  property OnRxChar: TGSMXOnRxChar (p.Chyba! Záložka není definována.) read write;
```

See TGSM (p.34).OnRxChar (p.40)

OnUnsolicitedLine (TGSMX)

```
published
  property OnUnsolicitedLine: TGSMXOnUnsolicitedLine (p.Chyba! Záložka není definována.) read write;
```

See TGSM (p.34).OnUnsolicitedLine (p.40)

OnUnsolicitedSMS (TGSMX)

```
published
  property OnUnsolicitedSMS: TGSMXOnUnsolicitedSMS (p.Chyba! Záložka není definována.) read write;
```

See TGSM (p.34).OnUnsolicitedSMS (p.40)

OnNetworkRegistration (TGSMX)

```
published
  property OnNetworkRegistration: TGSMXOnNetworkRegistration (p.Chyba! Záložka není definována.) reac
```

See TGSM (p.34).OnNetworkRegistration (p.41)

OnBusyChanged (TGSMX)

```
published
  property OnBusyChanged: TGSMXOnBusyChanged (p.Chyba! Záložka není definována.) read write;
```

See TGSM (p.34).OnBusyChanged (p.41)

IStrings2

```
type
  IStrings2 = interface( IStrings );
```

Extensions of IStrings interface that support objects associated to string (like TStrings.Objects)

Objects (IStrings2)

```
published
  property Objects[ Index: Integer ]: OleVariant read write;
```

Associated object array

ISMS

```
type
  ISMS = interface( IDispatch );
```

See TSMS (p.32)

ISMS2

```
type
  ISMS2 = interface( ISMS (p.50) );
```

See TSMS2 (p.32)

ISMSDeliver

```
type
  ISMSDeliver = interface( ISMS2 (p.50) );
```

See TSMSDeliver (p.33)

ISMSSubmit

```
type
  ISMSSubmit = interface( ISMS2 (p.50) );
```

See TSMSSubmit (p.33)

ISMSStatusReport

```
type
  ISMSStatusReport = interface( ISMS2 (p.50) );
```

See TSMSStatusReport (p.33)

ISMSCommand

```
type
  ISMSCommand = interface( ISMS2 (p.50) );
```

See TSMSCommand (p.33)

ISMSDeliverReport

```
type
  ISMSDeliverReport = interface( ISMS2 (p.50) );
```

See TSMSDeliverReport (p.33)

ISMSSubmitReport

```
type
  ISMSSubmitReport = interface( ISMSDeliverReport (p.51) );
```

See TSMSSubmitReport (p.34)

ICellBroadcast

```
type
  ICellBroadcast = interface( ISMS (p.50) );
```

See TCellBroadcast (p.34)

IGlobal

```
type
  IGlobal = interface( IDispatch );
```

Interface that provides global procedures declared in `GSM` (p.32) and `CommConnect` (p.13) units. To call the procedure create an `Global` object and appropriate procedure.
Note: I don't know how to declare global ActiveX procedures. It's possible implement into OCX classic DLL stdcall procedures but it does not work correctly with Visual Basic because no automation interface is exposed (Widestring/BSTR problems)

Unit ScktConnect

Unit defines classes for socket connection using `TCommunicationConnection` class

TClientSocketConnection

```
type
  TClientSocketConnection = class( TCommunicationConnection );
```

Object for connecting using TCP/IP socket to server. Object implements client socket side.

Address (TClientSocketConnection)

```
published
  property Address: string read write;
```

server address, see TCustomWinSocket

Host (TClientSocketConnection)

```
published
  property Host: string read write;
```

server host address, see TCustomWinSocket

Port (TClientSocketConnection)

```
published
  property Port: Integer read write;
```

see TCustomWinSocket

Service (TClientSocketConnection)

```
published
  property Service: string read write;
```

see TCustomWinSocket

Socket (TClientSocketConnection)

```
public
  property Socket: TCustomWinSocket read;
```

Communication thread instance

Unit TAPICom

TAPI objects encapsulate TAPI functionality. With the TAPI it is possible work as with a common `Windows.OpenFile` based device. The TAPI uses individual lines. The line is an installed TAPI device (modem, COM, etc.)
TAPI v-2.0 is supported (*Windows9x/NT*). Newer version v3.0 used in *Windows 2000* is not fully supported. Devices written for *TAPI 3.0* may not work.
Of course there is no compatibility with Kylix.

TTAPILine

```
type
  TTAPILine = class( TConnection );
```

Object initializes TAPI, configures TAPI, implements settings saving and restoring.

KeepConnection (TTAPILine)

```
published
  property KeepConnection: Boolean read write;
```

If the value is true TAPI remains opened after the last line was closed.

See also
`AddLine` (p.55) `RemoveLine` (p.55)

LineApp (TTAPILine)

```
public
  property LineApp: hLineApp read;
```

The TAPI handle

NumDevs (TTAPILine)

```
public
  property NumDevs: Integer read;
```

Returns number of TAPI lines

Count (TTAPILine)

```
public
  property Count: Integer read;
```

Returns number of added lines

Lines (TTAPILine)

```
public
  property Lines[ Index: Integer ]: TLineCom (p.56) read; default;
```

The list of added lines

LineNames (TTAPILine)

```
public
  property LineNames[ aDeviceId: DWord ]: string read;
```

Returns the name list of added lines.

APIVersions (TTAPILine)

```
public
  property APIVersions[ aDeviceId: DWord ]: DWord read;
```

Reads the version of the *aDeviceId* line.

DevConfig (TTAPILine)

```
public
  property DevConfig[ aDeviceId: DWord ]: string read write;
```

Reads/writes the configuration settings of the *aDeviceId* line. The returned configuration is binary and line-dependent.

AddLine (TTAPILine)

```
public
  procedure AddLine( aLine: TLineCom (p.56) );
```

Adds a line into TAPI the line list. TAPI is initialized when was added first line and will be closed when is removed the last line (except if `KeepConnection` (p.54) is `True`).

See also

RemoveLine

RemoveLine (TTAPILine)

```
public
  procedure RemoveLine( aLine: TLineCom (p.56) );
```

Removes a line from TAPI the line list. TAPI is initialized when was added first line and will be closed when is removed the last line (except if `KeepConnection` (p.54) is `True`).

See also

AddLine

ShowConfigDialog (TTAPILine)

```
public
  procedure ShowConfigDialog( aDeviceId: DWord );
```

Shows the standard line configuration dialog. The settings is possible save or restore by means of the `DevConfig` (p.55) property.

ShowTranslateDialog (TTAPILine)

```
public
  procedure ShowTranslateDialog(
      aDeviceId: DWord;
      aPhoneNumber: string );
```

Shows the standard line translate dialog (dial parameters). The *aPhoneNumber* argument is the modem phone number. It is necessary for proper dial prefix processing or pass empty string if no phone translation required.

FindDeviceId (TTAPILine)

```
public
  function FindDeviceId( const aDeviceName: string ): DWord;
```

Returns the line id by the *aDeviceName* argument.

TLineCom

```
type
  TLineCom = class( TConnection );
```

Object implements the TAPI *linexxxx* functions and implements a concrete connection, for ex. dialup.
For other here not documented properties see *TAPI documentation*.

PhoneNumber (TLineCom)

```
published
  property PhoneNumber: string read write;
```

The `TranslatedPhoneNumber` (p.56) property is adjusted value of the `PhoneNumber` according current location.

TranslatedPhoneNumber (TLineCom)

```
public
  property TranslatedPhoneNumber: string read;
```

The `TranslatedPhoneNumber` property is adjusted value of the `PhoneNumber` (p.56) according current location.

Comm (TLineCom)

```
public
  property Comm: TLineCommHandle (p.Chyba! Záložka není definována.) read;
```

This line handle can be used in the `TCommHandle` (p.13) object as the value of the `TCommHandle` (p.13).`hCommDev` (p.14) property.

Unit XSMS

The unit implements kinds of SMS protocols that enable streaming data longer than max. allowed length of SM (160/140 characters).

TSMSProtocol

```
type
  TSMSProtocol = class;
```

Implements common SM protocol based on fragmenting/concatenating.

Data (TSMSProtocol)

```
published
  property Data: TString read write;
```

Data content

Reference (TSMSProtocol)

```
published
  property Reference: LongInt read write;
```

Reference data number

FragmentCount (TSMSProtocol)

```
public
  property FragmentCount: Integer read;
```

Number of fragments

InsertedCount (TSMSProtocol)

```
public
  property InsertedCount: Integer read;
```

Number of inserted fragments

Fragments (TSMSProtocol)

```
public
  property Fragments[ Index: Integer ]: TString read;
```

Text content of particular fragments

Status (TSMSProtocol)

```
public
  property Status: TSMSProtocolStatus (p.81) read;
```

Status of inserted fragments. Determines if data are completely (all fragments) received. See `InsertFragment` (p.58)

Stamp (TSMSProtocol)

```
public
  property Stamp: TDateTime read;
```

Stamp when was inserted first fragment, see InsertFragment (p.58)

Clear (TSMSProtocol)

```
public
  procedure Clear; virtual;
```

Deletes all fragments and set Status (p.57) to empty.

InsertFragment (TSMSProtocol)

```
public
  function InsertFragment( const aFragment: TString ): TSMSProtocolInsertStatus (p.81); virt
```

Inserts fragment to former inserted fragments

result:

If is smsprOK check Status (p.57) to see if data has been completed.

TSMSProtocol2

```
type
  TSMSProtocol2 = class( TSMSProtocol (p.57) );
```

Common SMS protocol with port addressing.

SourceAddress (TSMSProtocol2)

```
published
  property SourceAddress: Word read write;
```

Address of source application port.

DestinationAddress (TSMSProtocol2)

```
published
  property DestinationAddress: Word read write;
```

Address of destination application port.

Options (TSMSProtocol2)

```
published
  property Options: TSMSProtocolOptionsSet (p.Chyba! Záložka není definována.) read write;
```

SMS protocol options.

TNBS

```
type
  TNBS = class( TSMSProtocol2 (p.58) );
```

The component implements narrow band socket protocol (NBS). NBS is the protocol used to send non-text content over SMS. NBS uses concatenated SMS messages and includes *port number* which is really a content type identifier, by which the phone recognize which kind of content is being sent to it.

OtherHeader (TNBS)

```
published
  property OtherHeader: TString read write;
```
Optional other headers of NBS protocol

Binary (TNBS)

```
published
  property Binary: Boolean read write;
```
If is `True` length of particular messages is 140, otherwise is 160.

TIE

```
type
  TIE = class;
```
Implements information element (IE) of `TUDHProtocol` (p.59)

IEI (TIE)

```
public
  property IEI: Byte read write;
```
Information element identifier

IED (TIE)

```
public
  property IED: TString read write;
```
Information element data

TUDHProtocol

```
type
  TUDHProtocol = class( TSMSProtocol2 (p.58) );
```
The class implements protocol using user data headers (UDH). In plain SM parameter UDHI indicating that the UD field contains a header

IEs (TUDHProtocol)

```
public
  property IEs: TObjectList read;
```
List of information elements (IE) of `TIE` (p.59) class

TSiemensOTA

```
type
  TSiemensOTA = class( TSMSProtocol (p.57) );
```
Starting with the S 45 and ME 45, Siemens mobile phones will provide over-the-air OTA) download capabilities for different types of content.
This component supports these capabilities according to OTA specification (release 1.0.1).

ObjectName (TSiemensOTA)

```
published
  property ObjectName: TString read write;
```

Is the name of the dataobject. If the file-extension is necessary for the processing in the mobile phone, it has to be provided with the object-name.

ObjectType (TSiemensOTA)

```
published
  property ObjectType: TString read write;
```

Is defining the type of the object (Bitmap: `'bmp'`, MIDI: `'mid'`)

TSMSProtocolStack

```
type
  TSMSProtocolStack = class( TStringList );
```

The class implements stack of all `TSMSProtocol` (p.57) packages. All incomming SM are processed by `ProcessSMS` (p.60) procedure and are pushed to relevant SMS protocol instance according originating phone number, reference number and content type (destination port).

ProcessSMS (TSMSProtocolStack)

```
public
  function ProcessSMS(
      aSMS: TSMS (p.32);
      var aId: TSMSProtocol (p.57) ): Boolean;
```

Processes incoming SM and try to find SM protocol instance that belonging to SM content

`aSMS`:
: received SM

`aId`:
: SMProtocol in which has been SM inserted

`result`:
: Return `True` if SM has been successfully inserted into a SM protocol (returned in aID parameter)

RemoveSMSProtocol (TSMSProtocolStack)

```
public
  procedure RemoveSMSProtocol( aId: TSMSProtocol (p.57) );
```

Removes SM protocol instance from stack

CleanSMSProtocols (TSMSProtocolStack)

```
public
  procedure CleanSMSProtocols(
      aRelativeDT: TDateTime;
      aOnlyNotComplete: Boolean );
```

Removes protocol instances from the stack that are older than *aRelativeDT* (and probably won't be completed)

`aRelativeDT`:
: how old SM protocols should be removed

`aOnlyNotComplete`:
Remove only not complete SM protocols

TDirectoryRecords

```
type
  TDirectoryRecords = class( TStringList );
```
Implements text MIME directory records

Example:

```
BEGIN:vCard
   VERSION:3.0
   FN:Tim Howes
   ORG:Netscape Communications Corp.
   ADR;TYPE=WORK:;;501 E. Middlefield Rd.;Mountain View;
    CA; 94043;U.S.A.
   TEL;TYPE=VOICE,MSG,WORK:+1-415-937-3419
   TEL;TYPE=FAX,WORK:+1-415-528-4164
   EMAIL;TYPE=INTERNET:howes@netscape.com
END:vCard
```

FoldingLength (TDirectoryRecords)

```
published
  property FoldingLength: Integer read write;
```
Number of max. line length. Longer line are wrapped using folding technique.

Data (TDirectoryRecords)

```
published
  property Data: TString read write;
```
Lists the fields in the TDirectoryRecords object as a single string with the individual strings delimited by carriage returns and line feeds and folded according `FoldingLength` (p.61).

Kind (TDirectoryRecords)

```
published
  property Kind: string read write;
```
Kind of directory record (for ex. 'VCARD').

Params (TDirectoryRecords)

```
public
  property Params[ Index: Integer ]: TStrings read;
```
List of parameters of field determined by index.

ParamsOf (TDirectoryRecords)

```
public
  property ParamsOf[ aName: string ]: TStrings read;
```
List of parameters of field determined by name.

TSmartMessage

```
type
  TSmartMessage = class;
```

The class implements Smart messaging according `Smert Message Specification revision 3.0.0`.
Smart messages are supported by Nokia phones.
To see what features are supported by selected phone see `Nokia_Phone_Messaging_Characteristics_v_1_3` document available at Nokia home site (http://www.forum.nokia.com > http://www.forum.nokia.com).
Most non-Nokia phones support EMS, see `TEMS` (p.72).

Data (TSmartMessage)

```
public
  property Data: TString read write;
```

Content of Smart message

CreateSM (TSmartMessage)

```
public
  class function CreateSM(
      const aData: TString;
      aPort: Word ): TSmartMessage (p.62);
```

Creates class instance according Smart message content and port number.

TSMMIMEDirectory

```
type
  TSMMIMEDirectory = class( TSmartMessage (p.62) );
```

Common class implementing directory features (`TSMvCard` (p.62), `TSMvCalendar` (p.63)) of Smart messages.

Records (TSMMIMEDirectory)

```
public
  property Records: TDirectoryRecords (p.61) read;
```

List of directory items

TSMvCard

```
type
  TSMvCard = class( TSMMIMEDirectory (p.62) );
```

Business card information transfer is based on the Versit vCard specification. The vCard specification defines a format for electronic business cards. This format is suitable to be used as an interchange format between applications or systems, and it is independent of the method used to transport it. See [RFC_2425], [RFC_2426].

TSMvCalendar

```
type
  TSMvCalendar = class( TSMMIMEDirectory (p.62) );
```

Calendar information transfer is based on the Versit vCalendar specification. The vCalendar specification defines a format for electronic calendaring and scheduling. This format is suitable to be used as an interchange format between applications or systems, and it is independent of the method used to transport it. The vCalendar enables exchange of event and to-do types of calendaring and scheduling events. An event represents a scheduled amount of time on a calendar, and a to-do item represents an action-item or assignment.

TSMOTABitmap

```
type
  TSMOTABitmap = class( TSmartMessage (p.62) );
```

The OTA bitmap format enables graphical information to be sent to a wide variety of handsets. Depending on the handset implementation, it may be possible for the user to create graphical objects and then send them to other handsets. Various applications can use this information to create a more illustrative and attractive outlook for the application.

InfoFieldCount (TSMOTABitmap)

```
public
  property InfoFieldCount: Byte read write;
```

Number of info fields

InfoFields (TSMOTABitmap)

```
public
  property InfoFields[ Index: Integer ]: Byte read write;
```

Content of info fields

AnimatedImageCount (TSMOTABitmap)

```
public
  property AnimatedImageCount: Integer read write;
```

Number of animations in OTA bitmap

Width (TSMOTABitmap)

```
public
  property Width: Integer read write;
```

Horizontal width of the bitmap in pixels.

Height (TSMOTABitmap)

```
public
  property Height: Integer read write;
```

Vertical height of the bitmap in pixels.

Images (TSMOTABitmap)

```
public
  property Images[ Index: Integer ]: TBitmap read write;
```

Bitmaps used in OTA bitmap. Bitmaps `1`..`AnimatedImageCount` (p.63)`-1` are animated bitmap. Bitmap `0` is mandatory.

ImportFromImage (TSMOTABitmap)

```
public
  procedure ImportFromImage( aBMP: TBitmap );
```

Imports image from bitmap according to `Width` (p.63), `Height` (p.63) and `AnimatedImageCount` (p.63).

ExportToImage (TSMOTABitmap)

```
public
  procedure ExportToImage( aBMP: TBitmap );
```

Imports image to bitmap according to `Width` (p.63), `Height` (p.63) and `AnimatedImageCount` (p.63).

TSMScreenSaver

```
type
  TSMScreenSaver = class( TSMOTABitmap (p.63) );
```

Implements screen saver features.

TSMCLIIcon

```
type
  TSMCLIIcon = class( TSMOTABitmap (p.63) );
```

The Calling Line Identification (CLI) icon is a bitmap, which can be attached to some number or numbers in the handset's phonebook (a caller group). When the caller is identified, the attached CLI icon is shown alongside other appropriate information such as the name and/or number of the caller. The CLI icon format doesn't contain any phonebook information so the linking between the phonebook entry and the CLI icon must be done in the handset.

TSMOperatorLogo

```
type
  TSMOperatorLogo = class( TSMOTABitmap (p.63) );
```

The Operator Logo is a bitmap, which can be shown alongside the operator identification when the display of the handset is in idle mode. The Operator Logo format contains operator identification information. It is up to handset implementation how to this information is used.

MCC (TSMOperatorLogo)

```
public
  property MCC: Word read write;
```

GSM Mobile Country Code

MNC (TSMOperatorLogo)

```
public
  property MNC: Byte read write;
```

GSM Mobile Network Code

TSMTextISO

```
type
  TSMTextISO = class( TSmartMessage (p.62) );
```

ISO text message used as part of TSMMultipartMessage (p.71)

Text (TSMTextISO)

```
public
  property Text: TString read write;
```

Text of SM

TSMTextUnicode

```
type
  TSMTextUnicode = class( TSmartMessage (p.62) );
```

UNICODE text message used as part of TSMMultipartMessage (p.71)

Text (TSMTextUnicode)

```
public
  property Text: WideString read write;
```

(Wide) text of SM

TSMProfile

```
type
  TSMProfile = class( TSMTextUnicode (p.65) );
```

Profile used as part of TSMMultipartMessage (p.71)

TBitStream

```
type
  TBitStream = class;
```

Auxiliary object used for bit streaming in TSMRingingTone (p.71) class

Data (TBitStream)

```
public
  property Data: TString read write;
```

Binary stream data

EOF (TBitStream)

```
public
  property EOF: Boolean read;
```

If end of stream is reached

PutFillers (TBitStream)

```
public
  procedure PutFillers;
```

Put bits to fill byte

SkipFillers (TBitStream)

```
public
  procedure SkipFillers;
```

Skip bits to end of byte

GetBit (TBitStream)

```
public
  function GetBit{}: Boolean;
```

Get one bit from stream

PutBit (TBitStream)

```
public
  procedure PutBit( aVal: Boolean );
```

Put one bit in stream

GetBits (TBitStream)

```
public
  function GetBits( aNum: Byte ): LongWord;
```

Get *aNum* bits from stream

PutBits (TBitStream)

```
public
  procedure PutBits(
      aVal: LongWord;
      aNum: Byte );
```

Put *aNum* bits in stream

Move (TBitStream)

```
public
  procedure Move( aShift: Integer );
```

Move current pointer up or down

TSMRTPattern

```
type
  TSMRTPattern = class;
```

Implements TSMRingingTone (p.71) pattern

Instructions (TSMRTPattern)

```
public
  property Instructions: TObjectList read;
```

List of instructions of TSMRTInstruction (p.67) type

Id (TSMRTPattern)

```
public
  Id: Byte;
```

see smrtpidPartx constants

Loop (TSMRTPattern)

```
public
  Loop: Byte;
```

Indicates how many times the pattern should be repeated. The value zero means no repeat. The value 255 means infinite.

TSMRTInstruction

```
type
  TSMRTInstruction = class;
```

Implements instruction features of TSMRTPattern (p.66)

CreateFromStream (TSMRTInstruction)

```
public
  class function CreateFromStream( BS: TBitStream (p.65) ): TSMRTInstruction (p.67);
```

Creates class instance from the bit stream

TSMRTNote

```
type
  TSMRTNote = class( TSMRTInstruction (p.67) );
```

Implements note features

Value (TSMRTNote)

```
public
  Value: Byte;
```

note value, see smrtiNotexxxx constants

Duration (TSMRTNote)

```
public
  Duration: Byte;
```

note duration, see smrtiDurationxxxx constants

DurationSpecifier (TSMRTNote)

```
public
  DurationSpecifier: Byte;
```
note duration specifier, see `smrtiDurationSpecxxxx` constants

TSMRTScale

```
type
  TSMRTScale = class( TSMRTInstruction (p.67) );
```
Implements note scale features

NoteScale (TSMRTScale)

```
public
  NoteScale: Byte;
```
see `smrtiScalexxxx`

TSMRTStyle

```
type
  TSMRTStyle = class( TSMRTInstruction (p.67) );
```
Implements style features

Value (TSMRTStyle)

```
public
  Value: Byte;
```
see `smrtiStylexxxx`

TSMRTTempo

```
type
  TSMRTTempo = class( TSMRTInstruction (p.67) );
```
Implements tempo features

DecodeBeatsPerMin (TSMRTTempo)

```
public
  class function DecodeBeatsPerMin( aBeats: Byte ): Word;
```
Converts encoded beats-per-min to plain value in real units.

EncodeBeatsPerMin (TSMRTTempo)

```
public
  class function EncodeBeatsPerMin( aBeats: Word ): Byte;
```
Converts plain value in real units to encoded beats-per-min value.

BeatsPerMin (TSMRTTempo)

```
public
  BeatsPerMin: Byte;
```

Beats per minute encoding, see DecodeBeatsPerMin (p.68), EncodeBeatsPerMin (p.68), smrtiTempoDefault

TSMRTVolume

```
type
  TSMRTVolume = class( TSMRTInstruction (p.67) );
```

Implements volume features

Value (TSMRTVolume)

```
public
  Value: Byte;
```

volume value, see smrtiVolumeDefault

TSMRTCommandPart

```
type
  TSMRTCommandPart = class;
```

Common command part of ringing tone (TSMRingingTone (p.71))

CreateFromStream (TSMRTCommandPart)

```
public
  class function CreateFromStream( BS: TBitStream (p.65) ): TSMRTCommandPart (p.69);
```

Creates class instance from bit stream

TSMRTRingingTone

```
type
  TSMRTRingingTone = class( TSMRTCommandPart (p.69) );
```

Defines name of ringing tone

TSMRTUnicode

```
type
  TSMRTUnicode = class( TSMRTCommandPart (p.69) );
```

Switches from ISO to UNICODE

TSMRTCancel

```
type
  TSMRTCancel = class( TSMRTCommandPart (p.69) );
```

Switches from UNICODE to ISO

TSMRTSound

```
type
  TSMRTSound = class( TSMRTCommandPart (p.69) );
```

Common song class

CreateFromStream (TSMRTSound)

```
public
  class function CreateFromStream( BS: TBitStream (p.65) ): TSMRTCommandPart (p.69);
```

Creates song instance from bit stream

TSMRTTemporarySong

```
type
  TSMRTTemporarySong = class( TSMRTSound (p.70) );
```

Pattern/instruction defined song

Patterns (TSMRTTemporarySong)

```
public
  property Patterns: TObjectList read;
```

List of patterns of TSMRTPattern (p.66) class

TSMRTMidiSong

```
type
  TSMRTMidiSong = class( TSMRTSound (p.70) );
```

MIDI song, not implemented

TSMRTDigitisedSong

```
type
  TSMRTDigitisedSong = class( TSMRTSound (p.70) );
```

Digitised song, not implemented

TSMRTPolyphonicSong

```
type
  TSMRTPolyphonicSong = class( TSMRTSound (p.70) );
```

Polyphonic song, not implemented

TSMRTCommand

```
type
  TSMRTCommand = class;
```

Implements commands of TSMRingingTone (p.71)

Parts (TSMRTCommand)

```
public
  property Parts: TObjectList read;
```

List of parts of TSMRTCommandPart (p.69)

TSMRingingTone

```
type
  TSMRingingTone = class( TSmartMessage (p.62) );
```

The ringing tone format enables ringing tones to be sent to a wide variety of handsets. Depending on the handset implementation, it may be possible for the user to create ringing tones and then send them to other handsets.
The ringing tone format is handset independent, and describes only the audio related information. It enables transmission of both basic songs and temporary songs. A basic song is intended to be saved in a handset while the temporary songs can be used together with an alert router to implement message notification with a special ringing tone.
See specification of *SMS 3.0.0*.

Commands (TSMRingingTone)

```
public
  property Commands: TObjectList read;
```

Commands parts of ringing tone

RTTTL (TSMRingingTone)

```
public
  property RTTTL: string read write;
```

Import/export to RTTTL (Ring Tone Text Transfer Language) format

TSMMultipartMessage

```
type
  TSMMultipartMessage = class( TSmartMessage (p.62) );
```

The Multipart Message format can be used for sending picture messages and whole profiles to mobile phones.
A Picture Message is a message format that consists of a picture and a text part. In the message, the text and the picture may be in either order (i.e. text part first, or picture first). Neither part is optional.
A Downloadable Profile is a message format that allows the sending of user profiles to mobile phones. A Downloadable Profile consists of a profile name, a 'screen saver' and a ringing tone. These three parts may be in any order, and all parts are optional.

Multiparts (TSMMultipartMessage)

```
public
  property Multiparts: TObjectList read;
```

List of parts that contain message. The possible parts are ISO/UNICODE text, OTA bitmap, ringing tone, profile, screen saver

TSMUnknown

```
type
  TSMUnknown = class( TSmartMessage (p.62) );
```

Smart message of an unknown type

TEMS

```
type
  TEMS = class;
```

he Enhanced Messaging Service (EMS) is based upon the standard SMS, but with formatting added to the text. The formatting may permit the message to contain animations, pictures, melodies, formatted text, and vCard and vCalendar objects. Objects may be mixed together into one message.
EMS are supported generally by Sony/Ericsson, Siemens and Alcatel phones. EMS are not supported by Nokia phones, they are using Smart messages(see `TSmartMessage` (p.62)).
See *Technical realization of the Short Message Service (SMS) (3GPP TS 23.040 V6.0.1)*.
EMS are transferred using `TUDHProtocol` (p.59).

Text (TEMS)

```
public
  property Text: TString read write;
```

Text of message

Objects (TEMS)

```
public
  property Objects: TObjectList read;
```

List of objects in EMS of `TEMSObject` class

ReadFrom (TEMS)

```
public
  procedure ReadFrom( aProtocol: TUDHProtocol (p.59) );
```

Reads EMS from SM protocol

WriteTo (TEMS)

```
public
  procedure WriteTo( aProtocol: TUDHProtocol (p.59) );
```

Writes EMS to SM protocol

TEMSObject

```
type
  TEMSObject = class;
```

Implements common EMS object that included into the SM text

IEI (TEMSObject)

```
public
  property IEI: Byte read;
```
Information element identifier

IED (TEMSObject)

```
public
  property IED: TString read write;
```
Information element data

CreateFromIEI (TEMSObject)

```
public
  class function CreateFromIEI( aIEI: Byte ): TEMSObject (p.72);
```
Creates class instance according information element identifier

TEMSPosObject

```
type
  TEMSPosObject = class( TEMSObject (p.72) );
```
Implements common EMS object that has position

Pos (TEMSPosObject)

```
published
  property Pos: Byte read write;
```
Position indicating in the SM data the instant the object shall be displayed or played in the SM data

TEMSUnknown

```
type
  TEMSUnknown = class( TEMSObject (p.72) );
```
An EMS object of unknown type

Data (TEMSUnknown)

```
published
  property Data: TString read write;
```
raw data of OBJECT

TEMSTextFormat

```
type
  TEMSTextFormat = class( TEMSPosObject (p.73) );
```
Implements EMS objects that affects formatting of SM text

Len (TEMSTextFormat)

```
published
  property Len: Byte read write;
```

Length of text that should be affected

Alignment (TEMSTextFormat)

```
published
  property Alignment: Byte read write;
```

see `emstxtAlignmentxxxx` constants

Size (TEMSTextFormat)

```
published
  property Size: Byte read write;
```

Size of text, see `emstxtSizexxxx`

Bold (TEMSTextFormat)

```
published
  property Bold: Boolean read write;
```

If text is bold

Italic (TEMSTextFormat)

```
published
  property Italic: Boolean read write;
```

If text is italic

Underlined (TEMSTextFormat)

```
published
  property Underlined: Boolean read write;
```

If text is underlined

StrikeThrough (TEMSTextFormat)

```
published
  property StrikeThrough: Boolean read write;
```

If text is strikethrough

ForegroundColor (TEMSTextFormat)

```
published
  property ForegroundColor: Byte read write;
```

Text foreground color, see `emsclxxxx` constants

BackgroundColor (TEMSTextFormat)

```
published
  property BackgroundColor: Byte read write;
```

Text background color, see `emsclxxxx` constants

TEMSSoundPredef

```
type
  TEMSSoundPredef = class( TEMSPosObject (p.73) );
```

There are a number of predefined sounds. These sounds are not transferred over the air interface, only the identification of them. There are 10 different sounds that can be added in the message, and as soon as the sound mark is in focus (on the display), the sound will be played.

SoundId (TEMSSoundPredef)

```
published
  property SoundId: Byte read write;
```

Identification of sound, see `emssndxxxx` constants

TEMSSoundUserDef

```
type
  TEMSSoundUserDef = class( TEMSPosObject (p.73) );
```

The sender can define own melodies according to the iMelody format. These melodies are transferred in the SM and can take up to 128 bytes.

Records (TEMSSoundUserDef)

```
published
  property Records: TDirectoryRecords (p.61) read;
```

iMelody definition, see *Infrared Data Association. Specifications for Ir Mobile Communications (IrMC) iMelody*)

TEMSAnimationPredef

```
type
  TEMSAnimationPredef = class( TEMSPosObject (p.73) );
```

There are number of predefined animations. These animations are not sent as animation over the air interface, only the identification of them. As soon as the position of the animation in the SM data is reached, the animation corresponding to the received number shall be displayed in a manner which is manufacturer specific.

AnimationId (TEMSAnimationPredef)

```
published
  property AnimationId: Byte read write;
```

identification of animation, see `emsanixxxx` constants

TEMSPicture

```
type
  TEMSPicture = class( TEMSPosObject (p.73) );
```

It is possible to include either a small (16**16 pixels), large (32**32 pixels) or pictures of variable size. These pictures have neither animation nor grey scale; they are plain black and white. All pictures are user defined.

Width (TEMSPicture)

```
published
  property Width: Integer read write;
```

horizontal size of a picture

Height (TEMSPicture)

```
published
  property Height: Integer read write;
```

vertical size of a picture

ImportFromImage (TEMSPicture)

```
public
  procedure ImportFromImage( aBMP: TBitmap );
```

Imports image from bitmap according Width (p.76) and Height (p.76)

TEMSPictureVariable

```
type
  TEMSPictureVariable = class( TEMSPicture (p.75) );
```

Is a picture of variable length

TEMSPictureSmall

```
type
  TEMSPictureSmall = class( TEMSPicture (p.75) );
```

Is a picture of size 16x16 pixels

TEMSPictureLarge

```
type
  TEMSPictureLarge = class( TEMSPicture (p.75) );
```

Is a picture of size 16x16 pixels

TEMSAnimation

```
type
  TEMSAnimation = class( TEMSPosObject (p.73) );
```

The user-defined animations consist of 4 pictures and there are two different sizes of these animations. The picture size of the small animations is 8**8 pixels and the large 16**16 pixels. These animations are sent over the air interface.

Width (TEMSAnimation)

```
published
  property Width: Integer read write;
```
Horizontal size of an animation

Height (TEMSAnimation)

```
published
  property Height: Integer read write;
```
Vertical size of an animation

Images (TEMSAnimation)

```
public
  property Images[ Index: Integer ]: TBitmap read write;
```
List of 4 bitmaps that create animation

ImportFromImage (TEMSAnimation)

```
public
  procedure ImportFromImage( aBMP: TBitmap );
```
Imports images from bitmap according Width (p.77) and Height (p.77)

ExportToImage (TEMSAnimation)

```
public
  procedure ExportToImage( aBMP: TBitmap );
```
Exports four images to bitmap according Width (p.77) and Height (p.77)

TEMSAnimationSmall

```
type
  TEMSAnimationSmall = class( TEMSAnimation (p.76) );
```
Is an animation of size 8x8 pixels

TEMSAnimationLarge

```
type
  TEMSAnimationLarge = class( TEMSAnimation (p.76) );
```
Is an animation of size 16x16 pixels

TEMSUserPrompt

```
type
  TEMSUserPrompt = class( TEMSObject (p.72) );
```
With the User Prompt Indicator a sending entity is able to indicate to the receiving entity, that the following object is intended to be handled at the time of reception, e.g. by means of user interaction. The object may be a picture, an animation, a User Defined Sound or a combination of these.

NumOfObjects (TEMSUserPrompt)

```
public
  property NumOfObjects: Byte read write;
```
Number of corresponding objects

TEMSObjectDistributionIndicator

```
type
  TEMSObjectDistributionIndicator = class( TEMSObject (p.72) );
```
This facility allows a level of control to be requested over the distribution of objects contained within selected information elements in short messages.
If no Object Distribution Indicator is specified for an information element in which an object is received, then that object may be freely distributed

NumOfIE (TEMSObjectDistributionIndicator)

```
public
  property NumOfIE: Byte read write;
```
This octet specifies the number of information elements from 1-255 for which the Distribution Attributes in the next octet shall apply. The affected objects shall be contained in Information Elements immediately following this IE and may be contained in subsequent short message segments within a concatenated short message.

Attrib (TEMSObjectDistributionIndicator)

```
public
  property Attrib: Byte read write;
```
see `emsodiaxxxx` constants

TEMSXObject

```
type
  TEMSXObject = class( TEMSObject (p.72) );
```
The Extended Object allows an extended code range for format types. The Extended Object may extend across segment boundaries of a concatenated short message. A single segment may include one or more Extended Object IEs.

HeaderFlag (TEMSXObject)

```
published
  property HeaderFlag: Boolean read write;
```
Include extended object header when encoding/decoding

Reference (TEMSXObject)

```
published
  property Reference: Byte read write;
```
A modulo 256 counter indicating the reference number for the Extended Object. Two different Extended Objects in a single concatenated message shall have different reference numbers.

Pos (TEMSXObject)

```
published
  property Pos: Word read write;
```

The Extended Object Position indicates the absolute character position within the message text after which the object shall be played or displayed. The absolute character position relates to the entire text within the concatenated message, the first character is numbered character 1.

Attrib (TEMSXObject)

```
published
  property Attrib: Byte read write;
```

control byte, see `emsxaxxxx` constants

DataLength (TEMSXObject)

```
published
  property DataLength: Word read write;
```

Length of data

Kind (TEMSXObject)

```
published
  property Kind: Byte read write;
```

This octet indicates the format of the Extended Object

Data (TEMSXObject)

```
published
  property Data: TString read write;
```

Extended Object Data

TEMSXObjectReused

```
type
  TEMSXObjectReused = class( TEMSObject (p.72) );
```

his facility is used to reuse an Extended Object in a message which has already been defined in the same message

Reference (TEMSXObjectReused)

```
published
  property Reference: Byte read write;
```

Reference number of the Extended Object to be reused

Pos (TEMSXObjectReused)

```
published
  property Pos: Word read write;
```

indicates in the concatenated message the absolute character position after which the object shall be played or displayed

TEMSXObjectDataRequest

```
type
  TEMSXObjectDataRequest = class( TEMSObject (p.72) );
```

Upon receiving this IE in an SMS-DELIVER PDU, if an MS supports this request and the corresponding response, it shall respond with an SMS-DELIVER-REPORT PDU containing a Data Format Delivery Request as defined in the Extended Object IE. This SMS-DELIVER PDU may be discarded.

TEMSWVGObject

```
type
  TEMSWVGObject = class( TEMSPosObject (p.73) );
```

A message may contain one or more Wireless Vector Graphics (WVG) objects. A WVG object is a vector graphics picture or animation and is scalable. Two subtypes of WVG objects are supported; Standard WVG object and Character Size WVG object. Actual display size of a Standard WVG object depends on display screen size and MMI implementation on terminals. A Character Size WVG object has a height that equals or is similar to the height of message text but with variable width. Character Size WVG object may be edited in the same way as standard text, e.g. insertion deletion and text wrapping.

Data (TEMSWVGObject)

```
published
  property Data: TString read write;
```

The WVG element is used to describe vector graphics objects. The vector graphics format is used to allow the creation of small pictures which may include simple animation or the creation small handwritten sketches. WVG makes use of the graphical primitives. These primitives can be used to describe a compact drawing.

TEMSWVGObjectStandard

```
type
  TEMSWVGObjectStandard = class( TEMSWVGObject (p.80) );
```

A Standard WVG object may or may not have fixed size. In either case, display size should be determined by the terminal implementation. Recommended display size is a largest possible size on terminal screen while aspect ratio shall be maintained.

TEMSWVGObjectCharSize

```
type
  TEMSWVGObjectCharSize = class( TEMSWVGObject (p.80) );
```

A Character Size WVG object is a small graphics similar to the size of a typed character. The display height for a Character Size WVG object is decided by the terminal implementation. Recommended Character Size WVG object height is to be similar to the message text font height. The width of a Character Size WVG object is variable depending on the aspect ratio defined in the object. Character Size WVG objects can appear more than one time in one message..

TEMSCompressionControl

```
type
  TEMSCompressionControl = class( TEMSUnknown (p.73) );
```

This information element is used to indicate a compressed octet sequence. The compression control is only used in association with one or more Extended Objects and/or Reused Extended Objects.

TSMSProtocolStatus

```
type
  TSMSProtocolStatus =
    ( smspsEmpty
    , smspsOK
    , smspsNotComplete
    );
```

Status of `TSMSProtocol` (p.57)

`smspsEmpty`:
No fragment in `TSMSProtocol` (p.57)

`smspsOK`:
All fragments has been inserted and data are available in `TSMSProtocol` (p.57).`Data` (p.57)

`smspsNotComplete`:
Not all fragments have been inserted, `TSMSProtocol` (p.57).`Data` (p.57) are incomplete

TSMSProtocolInsertStatus

```
type
  TSMSProtocolInsertStatus =
    ( smsprOK
    , smsprBadFragment
    , smsprFragmentDoesNotFit
    , smsprFragmentDifferentData
    );
```

Result of `TSMSProtocol` (p.57).`InsertFragment` (p.58)

`smsprOK`:
Inserted fragment has been sucesfully inserted

`smsprBadFragment`:
Inserted fragment has bad format.

`smsprFragmentDoesNotFit`:
Inserted fragment does not fit. It's probably protocol from other data package

`smsprFragmentDifferentData`:
Inserted fragment does fit but content differs from formerly inserted fragment

TSMSProtocolOptions

```
type
  TSMSProtocolOptions =
    ( smspoAddressing
    , smspoReference
    );
```

Options used for fragment encoding

`smspoAddressing`:

Force inserting address to fragments

`smspoReference`:

Force inserting address to fragments

smCommonTelChar

```
const
  smCommonTelChar = [ '-', '#', '*', 'W', 'w', 'P', 'p', smSpace (p.Chyba! Záložka není definována.)
```

Legal chars for phone number

smBlinkOn

```
const
  smBlinkOn = TChar( 'y' );
```

Enable blinking char

smBlinkOff

```
const
  smBlinkOff = TChar( 'o' );
```

Disable blinking char

emsodiaNoForwarding

```
const
  emsodiaNoForwarding = '01';
```

the associated object(s) shall not be forwarded by SMS

emsxaNoForwarding

```
const
  emsxaNoForwarding = '01';
```

object shall not be forwarded by SMS

emsxaUserPrompt

```
const
  emsxaUserPrompt = '02';
```

object shall be handled as a User Prompt