ZylGSM 1.53

ZylGSM is a Delphi & C++Builder component that communicates with a GSM modem.
Main features:
- send SMS in text mode
- send SMS in PDU mode
- delete SMS
- dial in voice mode
- dial in data mode
- answer incoming call
- hang up current conversation
- recognize calls and identify caller number
- new SMS message alert
- read PDU and text SMS
- GSM 7-bit alphabet and UCS2 support
- concatenated (long) SMS support
- detect GSM modem
- execute custom AT commands

This component works with AT (Hayes) compilant GSM modems connected to one of the serial ports. You can use it also with USB and bluetooth devices, because these devices have a driver that redirects the input from the USB or bluetooth port to a virtual serial port (you can check it in System/Device Manager/Modems).
- the message text must be in GSM 7-bit charset, GSM 8-bit charset or Unicode, concatenated SMS is also supported

The demo version is fully functional in Delphi and C++Builder IDE, but it displays a nag dialog (the licensed version will, of course, not have a nag dialog and will not be limited to the IDE). The package includes demo programs for Delphi and C++Builder and a help file with the description of the component.

Supported Operating Systems: 2000/XP/Serv2003/Vista/Serv2008/7/8/Serv2012/10/11
Available for: Delphi 12 Athens (Win32 & Win64), Delphi 11 Alexandria (Win32 & Win64), Delphi 10.4 Sydney (Win32 & Win64), Delphi 10.3 (Win32 & Win64), Delphi 10.2 (Win32 & Win64), Delphi 10.1 (Win32 & Win64), Delphi 10 (Win32 & Win64), Delphi XE8 (Win32 & Win64), Delphi XE7 (Win32 & Win64), Delphi XE6 (Win32 & Win64), Delphi XE5 (Win32 & Win64), Delphi XE4 (Win32 & Win64), Delphi XE3 (Win32 & Win64), Delphi XE2 (Win32 & Win64), Delphi XE, Delphi 2010, Delphi 2009, Delphi 2007, Delphi 2006, Delphi 7, Delphi 6, Delphi 5, C++Builder 12 Athens (Win32 & Win64), C++Builder 11 Alexandria (Win32 & Win64), C++Builder 10.4 Sydney (Win32 & Win64), C++Builder 10.3 (Win32 & Win64), C++Builder 10.2 (Win32 & Win64), C++Builder 10.1 (Win32 & Win64), C++Builder 10 (Win32 & Win64),

C++Builder XE8 (Win32 & Win64), C++Builder XE7, C++Builder XE6, C++Builder XE5, C++Builder XE4, C++Builder XE3, C++Builder XE2, C++Builder XE, C++Builder 2010, C++Builder 2009, C++Builder 2007, C++Builder 2006, C++Builder 6, Turbo Delphi, Turbo C++

Remarks:

- The Delphi 2006 version is fully compatible with Turbo Delphi
- The C++Builder 2006 version is fully compatible with Turbo C++

Insatallation:

If you have a previous version of the component installed, you must remove it completely before installing this version. To remove a previous installation, proceed as follows:
-Start the IDE, open the packages page by selecting Component - Install Packages
-Select ZylGSMPack package in the list and click the Remove button
-Open Tools - Environment Options - Library and remove the library path pointing to ZylGSM folder
-Close the IDE
-Browse to the folder where your bpl and dcp files are located (default is $(DELPHI)\Projects\Bpl for Delphi, $(BCB)\Projects\Bpl for C++ Builder). -Delete all of the files related to ZylGSM
-Delete or rename the top folder where ZylGSM is installed
-Start regedit (click Start - Run, type "regedit.exe" and hit Enter). Open the key HKEY_CURRENT_USER\Software\Borland\<compiler>\<version>\Palette and delete all name/value items in the list related to ZylGSM. (<compiler> is either "Delphi" or "C++Builder", <version> is the IDE version you have installed)

-Unzip the zip file and open the ZylGSMPack.dpk file in Delphi (ZylGSMPack.bpk file in C++Builder), compile and install it
and add to Tools/Environment Options/Library (in older Delphi/C++Builder menu) or Tools/Options/Delphi Options/Library/Library Path (in newer Delhi menu) or Tools/Options/C++ Options/Paths and Directories/Library Path & Include Path (in newer C++Builder menu) the path of the installation (where the ZylGSM.dcu file is located). The component will be added to the "Zyl Soft" tab of the component palette. After you have the component on your component palette, you can drag and drop it to any form, where you can set its properties by the Object Inspector and you can write event handlers selecting the Events tab of the Object Inspector and double clicking the preferred event.
If you still have problems in C++Builder, running an application, which contains the component, then open the project and in C++Builder menu, Project/Options/Packages and uncheck "Build with runtime packages".

-It is indicated to use this component with "Stop on Delphi exception" option deactivated. You can do this from Delphi / C++Builder menu, Tools/Debugger Options/Language Exceptions/Stop on Delphi exceptions in older versions or Tools/Options/Debugger Options/Embarcadero Debuggers/Language Exceptions/Notify on language exceptions in newer versions, otherwise you will have a break at all the handled exceptions.

Types:

TCommPort = (
spNone, spCOM1, spCOM2, spCOM3, spCOM4, spCOM5, spCOM6, spCOM7, spCOM8, spCOM9, spCOM10,
spCOM11, spCOM12, spCOM13, spCOM14, spCOM15, spCOM16, spCOM17, spCOM18, spCOM19, spCOM20,
spCOM21, spCOM22, spCOM23, spCOM24, spCOM25, spCOM26, spCOM27, spCOM28, spCOM29, spCOM30,
spCOM31, spCOM32, spCOM33, spCOM34, spCOM35, spCOM36, spCOM37, spCOM38, spCOM39, spCOM40,
spCOM41, spCOM42, spCOM43, spCOM44, spCOM45, spCOM46, spCOM47, spCOM48, spCOM49, spCOM50,

spCOM51, spCOM52, spCOM53, spCOM54, spCOM55, spCOM56, spCOM57, spCOM58, spCOM59, spCOM60,
spCOM61, spCOM62, spCOM63, spCOM64, spCOM65, spCOM66, spCOM67, spCOM68, spCOM69, spCOM70,
spCOM71, spCOM72, spCOM73, spCOM74, spCOM75, spCOM76, spCOM77, spCOM78, spCOM79, spCOM80,
spCOM81, spCOM82, spCOM83, spCOM84, spCOM85, spCOM86, spCOM87, spCOM88, spCOM89, spCOM90,
spCOM91, spCOM92, spCOM93, spCOM94, spCOM95, spCOM96, spCOM97, spCOM98, spCOM99, spCOM100,
spCOM101, spCOM102, spCOM103, spCOM104, spCOM105, spCOM106, spCOM107, spCOM108, spCOM109, spCOM110,
spCOM111, spCOM112, spCOM113, spCOM114, spCOM115, spCOM116, spCOM117, spCOM118, spCOM119, spCOM120,
spCOM121, spCOM122, spCOM123, spCOM124, spCOM125, spCOM126, spCOM127, spCOM128, spCOM129, spCOM130,
spCOM131, spCOM132, spCOM133, spCOM134, spCOM135, spCOM136, spCOM137, spCOM138, spCOM139, spCOM140,
spCOM141, spCOM142, spCOM143, spCOM144, spCOM145, spCOM146, spCOM147, spCOM148, spCOM149, spCOM150,
spCOM151, spCOM152, spCOM153, spCOM154, spCOM155, spCOM156, spCOM157, spCOM158, spCOM159, spCOM160,
spCOM161, spCOM162, spCOM163, spCOM164, spCOM165, spCOM166, spCOM167, spCOM168, spCOM169, spCOM170,
spCOM171, spCOM172, spCOM173, spCOM174, spCOM175, spCOM176, spCOM177, spCOM178, spCOM179, spCOM180,
spCOM181, spCOM182, spCOM183, spCOM184, spCOM185, spCOM186, spCOM187, spCOM188, spCOM189, spCOM190,
spCOM191, spCOM192, spCOM193, spCOM194, spCOM195, spCOM196, spCOM197, spCOM198, spCOM199, spCOM200,
spCOM201, spCOM202, spCOM203, spCOM204, spCOM205, spCOM206, spCOM207, spCOM208, spCOM209, spCOM210,
spCOM211, spCOM212, spCOM213, spCOM214, spCOM215, spCOM216, spCOM217, spCOM218, spCOM219, spCOM220,
spCOM221, spCOM222, spCOM223, spCOM224, spCOM225, spCOM226, spCOM227, spCOM228, spCOM229, spCOM230,
spCOM231, spCOM232, spCOM233, spCOM234, spCOM235, spCOM236, spCOM237, spCOM238, spCOM239, spCOM240,
spCOM241, spCOM242, spCOM243, spCOM244, spCOM245, spCOM246, spCOM247, spCOM248, spCOM249, spCOM250,
spCOM251, spCOM252, spCOM253, spCOM254, spCOM254, spCustom);

TBaudRate = (br000075, br000110, br000134, br000150, br000300, br000600, br001200, br001800,
br002400, br004800, br007200, br009600, br014400, br019200, br038400, br057600, br115200, br128000, br230400, br256000, br460800, br921600, brCustom);

TStopBits = (sb1Bit, sb1_5Bits, sb2Bits);
TDataWidth = (dw5Bits, dw6Bits, dw7Bits, dw8Bits);
TParityBits = (pbNone, pbOdd, pbEven, pbMark, pbSpace);
THwFlowControl = (hfNONE, hfDTRDSR, hfRTSCTS);
TSwFlowControl = (sfNONE, sfXONXOFF);
TCallType = (ctVoice, ctData);
TMessageFormat = (mfPDU, mfText);
TMessageEncoding = (meAuto, me7bit, me8bit, meUCS2);

TDetectionMethod = (dmStandard, dmSendSms, dmReadSms);
TConnectEvent = procedure(Sender: TObject; Port: TCommPort) of object;
TSendReceiveEvent = procedure(Sender: TObject; Buffer: AnsiString) of object;
TRingEvent = procedure(Sender: TObject; CallerNumber: AnsiString) of object;
TNewMessageEvent = procedure(Sender: TObject; Location: AnsiString; Index: Integer) of object;
TReadMessageEvent = procedure(Sender: TObject; MessageText: WideString; PhoneNumber, CenterNumber: AnsiString; TimeStamp: TDateTime; TimeZone, Status: Integer) of object;
EZylGSMException = class(Exception);

TDeviceSetting = class
property CommPort: TCommPort read FCommPort write FCommPort;
property BaudRate: TBaudRate read FBaudRate write FBaudRate;
constructor Create(); overload;
constructor Create(commPort: TCommPort; baudRate: TBaudRate); overload;
end;

TDeviceSettings = class
property Count: Integer read GetCount;
property Items[Index: Integer]: TDeviceSetting read GetItem write SetItem; default;
constructor Create();
destructor Destroy; override;
function Add(const Item: TDeviceSetting): Integer;
procedure Delete(Idx: Integer);
procedure Clear;
end;

TDetectionMethod = (dmStandard, dmSendSms, dmReadSms);

Properties:
Port: TCommPort - serial port name. If you change the port, you need to call the method Open again.
BaudRate: TBaudRate - baud rate value at which the communication device operates
CustomPortName: AnsiString - custom port name, when Port property is set to spCustom. Now you can open ports with any name.
CustomBaudRate - baud rate value at which the communication device operates, when BaudRate property is set to brCustom
DataWidth: TDataWidth - number of bits in the bytes transmitted and received
StopBits: TStopBits - number of stop bits to be used
Parity: TParityBits - parity scheme to be used
EnableDTROnOpen: Boolean - enable / disable DTR when the port is open
EnableRTSOnOpen: Boolean - enable / disable RTS when the port is open
HwFlowControl: THwFlowControl - hardware flow control
SwFlowControl: TSwFlowControl - software flow control
NeedSynchronization: Boolean - set this property to true for thread safety. If you use the component in ActiveX environment, set this property to false. The default value is false.
Priority: TThreadPriority - priority of the reader thread
AutoReadNewMessage: Boolean - if this property is true (default value), then new incoming messages are read automatically and after this, their status will be set to "received read message" (OnReadMessage event).
MessageFormat: TMessageFormat - preferred SMS message format
PIN: AnsiString - PIN code, if exists. If there is no PIN, leave it empty.
IsFaulted: Boolean - indicates that the last connection was faulted.
IsListing: Boolean - true, if the stored messages are listing, false, if already done. Use it with GetAllSMS method, to be able to determine when all the messages were read.
SkipIfSetupFailed: Boolean - when this property is true, if setting serial parameters like DCB in the

Open method fails, will ignore this.
CloseWhenLineStatusIsZero: Boolean - when this property is true and line status is empty, the port will be closed automatically, if line status was not empty, after you opened the port.
AutoReconnect: Boolean - Set this property to true, if you want to automatically reconnect to the serial port after a OnFault event, when the port is available again. Set AutoReconnect to true, before the port is faulted, otherwise it will have no effect.
The default value is false.
AutoReconnectCheckInterval: Integer - The time interval in milliseconds the serial port is trying to periodically reconnect, after a OnFault event occur, if AutoReconnect is set to true. It must be a positive value. The default value is 4000.
DetectionMethod: TDetectionMethod - The method how the GSM modem is detected.
ConnectionTimeout: **Cardinal** (milliseconds) - you can set a time-out value to automatically close the connection and fire the OnTimeout event, if there is no data received several seconds. A value of zero indicates that time-out is not used.
IdleInterval: Cardinal (milliseconds) - you can set an idle interval value to automatically to fire the OnIdle event, if there is no data received several seconds. A value of zero indicates that idle is not used.
IsIdle: Boolean - true, when the connection is in idle state.
StorageLocation: String - SMS storage location. Can be SIM or phone memory.

Methods - Connectivity:

function BaudRateToInt(pBaudRate: TBaudRate): Integer - converts TBaudRate type to integer
constructor Create(AOwner: TComponent) - constructor
destructor Destroy - destructor
procedure Open(skipInit: Boolean = False) - starts the communication. If skipInit is true, modem initialization will not execute, but you will have to call InitGSMModem method from code later on to work properly.
procedure Close - stops the communication
function IntToBaudRate(Value: Integer): TBaudRate - converts integer to TBaudRate type
function IsConnected: TCommPort - returns the comm port where the component is connected to
function StringToCommPort(Port: AnsiString): TCommPort - converts String to TCommPort
function CommPortToString(Port: TCommPort): AnsiString - converts TCommPort to String
function GetExistingCommPorts: TCommPortSet - returns the existing serial ports of the system
procedure GetExistingCommPortNames(Strings: TStrings) - returns the existing port names in Strings list parameter
function IsExistingCommPort(pport: String): Boolean - returns true if pport exists
procedure ResetIdleState - resets the idle state of the port.

Methods - GSM:

function DetectGSM(const startBaudRate, endBaudRate: TBaudRate; var pPort: TCommPort; var pBaudRate: TBaudRate): Boolean - detects the GSM modem connected to the system, returns as output parameters the communication port and baud rate. Only baud rate values between startBaudRate and endBaudRate will be checked.
function DetectGSM(var pPort: TCommPort; var pBaudRate: TBaudRate): Boolean - detects the first GSM modem connected to the system, returns as output parameters the communication port and baud rate.
function DetectGSM(startPort: TCommPort; const startBaudRate, endBaudRate: TBaudRate; var pPort: TCommPort; var pBaudRate: TBaudRate): Boolean - detects the first GSM modem connected to the system, returns as output parameters the communication port and baud rate. Only ports starting with startPort and baud rate values between startBaudRate and endBaudRate will be checked.
function FastDetectGSM(var pPort: String; var pBaudRate: TBaudRate): Boolean - detects the first GSM modem connected to the system, returns as output parameters the communication port and baud rate. Only baud rate values higher than 4800 will be checked.
function DetectAllGSM(startPort: TCommPort; const startBaudRate, endBaudRate:

TBaudRate): TDeviceSettings - detects all the GSM modems connected to the system. Only ports starting with startPort and baud rate values between startBaudRate and endBaudRate will be checked.
function DetectAllGSM(): TDeviceSettings - detects all the GSM modems connected to the system.
function FastDetectAllGSM(): TDeviceSettings - detects all the GSM modems connected to the system. Only baud rate values higher than 4800 will be checked.
function DialData(PhoneNumber: AnsiString): Boolean - dials PhonNumber in data mode. Returns true if succeeded.
function DialVoice(PhoneNumber: AnsiString): Boolean - dials PhonNumber in Voice mode. Returns true if succeeded.
function AnswerCall(): Boolean - answers incoming call. Returns true if succeeded.
function TerminateCall(): Boolean - hangs up current call. Returns true if succeeded.
function TestGSM(): Boolean - tests if the connection to the gsm modem is alive.
function SendSMSAsText(PhoneNumber, CenterNumber, MessageText: AnsiString): Boolean - sends SMS in text mode, if it's supported by the GSM modem. MessageText must be in GSM 7-bit charset.
function SendSMSAsPDU(PhoneNumber, CenterNumber: AnsiString; MessageText: WideString; Encoding: TMessageEncoding = meAuto; Flash: Boolean = false): Boolean - sends SMS in PDU mode. Unicode is supported. Concatenated SMS is supported. If Flash is true, then a class 0 sms will be sent.
function SendString(str: AnsiString): DWORD - sends a custom string to the GSM modem and returns the number of sent bytes.
function SetDefaultStorage(Location: AnsiString): Boolean - sets the default SMS storage (Phone_SIM or Phone_MEMORY).
procedure GetSMS(Location: AnsiString; Index: Integer) - returns the SMS from Location memory (**Phone_SIM** or **Phone_MEMORY**), indicated by Index. If the message exists, OnReadMessage event will fire with the text of the message, time stamp, caller and center number.
procedure GetAllSMS(Location: AnsiString; SmsType: Integer = 0) - returns all the SMS from Location memory (**Phone_SIM** or **Phone_MEMORY**), depending on SmsType, which can be:
0 - Received unread message, that is, new message. Default value
1 - Received read message
2 - Stored unsent message.
3 - Stored sent message.
4 - All messages.
OnReadMessage event will fire for all the messages with the text of the message, caller and center number.
procedure InitGSMModem() - initalizes the GSM modem. Normally you don't have to call it, because it is called automatically from method Open, if param skipInit is not true.
function DeleteSMS(Location: AnsiString; Index: Integer): Boolean - deletes an SMS message specified by Index from the selected Location (**Phone_SIM** or **Phone_MEMORY**). Special values of parameter Index:
-1 - delete all read messages from preferred message storage leaving unread messages and stored mobile originated messages (whether sent or not) untouched.
-2 - delete all read messages from preferred message storage and sent mobile originated messages, leaving unread messages and unsent mobile originated messages untouched.
-3 - delete all read messages from preferred message storage and sent mobile originated messages, leaving unread messages untouched.
-4 - delete all messages from preferred message storage including unread messages.
function GetIMEI: AnsiString - returns the IMEI number of the phone.
function GetPhoneNumber: AnsiString - returns the phone number of the SIM.
function GetSignalStrength: Integer - returns signal strength as follows:
Value RSSI dBm Condition
2 -109 Marginal
3 -107 Marginal

4 -105 Marginal
5 -103 Marginal
6 -101 Marginal
7 -99 Marginal
8 -97 Marginal
9 -95 Marginal
10 -93 OK
11 -91 OK
12 -89 OK
13 -87 OK
14 -85 OK
15 -83 Good
16 -81 Good
17 -79 Good
18 -77 Good
19 -75 Good
20 -73 Excellent
21 -71 Excellent
22 -69 Excellent
23 -67 Excellent
24 -65 Excellent
25 -63 Excellent
26 -61 Excellent
27 -59 Excellent
28 -57 Excellent
29 -55 Excellent
30 -53 Excellent
99 - not known or not detectable

function HideMyPhoneNumber(hideNumber: Boolean = True): Boolean - hides / shows phone number.Returns true if succeeded.

function SetAutoAnswer(noRings: Integer = 0): Boolean - if noRings is 0, then auto-answer for incoming calls is disabled, otherwise auto-answer is active after a number of rings specified by noRings parameter. Returns true if succeeded.

function SetEcho(state: Boolean): Boolean - sets if echo characters are received. Returns true if succeeded.

function EnterPIN(strPin: AnsiString): Boolean - enters the PIN code. Returns true if succeeded.

function ExecuteATCommand(strAT: AnsiString; ResultList: TStrings; CommandTerminator: AnsiChar = #13): Integer - executes any AT commands specified in the parameter strAT. The results will be sotred in the ResultList. Usually the first item of the list is the AT command, the second is the answer (result) and the third is the status: OK or ERROR. The returned value is the number of items of the ResultList.

function ExecuteATCommand(strAT: AnsiString; CommandTerminator: AnsiChar = #13): Boolean - executes any AT commands specified in the parameter strAT. Returns true if succeeded.

Example:

```
function TZylGSM.GetIMEI: AnsiString;
var
sl: TStringList;
begin
Result := '';
sl := TStringList.Create;
if ExecuteATCommand('AT+CGSN', sl) > 1 then
begin
Result := sl[1];
end;
sl.Free;
```

end;
function TestGSMModemCapabilities(): Word - tests the capability of the GSM modem.
1st bit - reserved
2nd bit - display caller number support (1 =true, 0 = false)
3rd bit - default GSM 7-bit alphabet support
4th bit - Text SMS support
5th bit - PDU SMS support
6th bit - new message notification support
7th bit - read SMS support
8th bit - send SMS support
9th -16th bit - reserved
procedure LoadPhoneBook(Location: AnsiString) - loads the contacts from the specified location into one of the phone book lists, SimPhoneBook or MemoryPhoneBook.
procedure AddContact(location, number, name: AnsiString) - adds a new contact to the phone book.
procedure DeleteContact(location: AnsiString; index: Integer) - deletes the contact with index from the phone book.
function EnableEcho(enable: Boolean): Boolean - enable / disable GSM modem echo. Resturns true, if suceeded.
function ReadBuffer(delay: Integer = 0): AnsiString - use this method to synchronously read incoming data
function ReadStringUpToEndChars(endChars1, endChars2: AnsiString; timeOutSeconds: Integer; var endCharsFound: AnsiString): AnsiString - synchronously reads the serial port, till one of the endChars sequence appears. endChars is not included in the result. If the ellapsed time is higher than timeout, then it will return with empty result (antiblocking protection). If timeout is 0, then there is no antiblocking protection. timedOut indicates if the read process was timed out. partialResult is empty string, if endChars sequence was found, otherwise contains the discarded data. endCharsFound is the terminator characters sequence, which was reached.
function ReadStringUpToEndChars(endCharsList: TStrings; timeoutSeconds: Integer; var partialResult: AnsiString; var timedOut: Boolean; var endCharsFound: AnsiString): AnsiString - synchronously reads the serial port, till one of the endChars sequence appears (one element of endCharsList). endChars is not included in the result. If the ellapsed time is higher than timeout, then it will return with empty result (antiblocking protection). If timeout is 0, then there is no antiblocking protection. timedOut indicates if the read process was timed out.
partialResult is empty string, if endChars sequence was found, otherwise contains the discarded data. endCharsFound is the terminator characters sequence, which was reached.

Events:

OnRing: TRingEvent=procedure(Sender: TObject; CallerNumber: AnsiString) **-** fires when there is an incoming call. The parameter CallerNumber contains the phone number of the caller.
OnReceive: TSendReceiveEvent=procedure(Sender: TObject; Buffer: AnsiString) - fires when new data was received. The parameter Buffer contains the received data.
OnSend: TSendReceiveEvent=procedure(Sender: TObject; Buffer: AnsiString) - fires after new data was sent. The parameter Buffer contains the sent data.
OnConnect: TConnectEvent=procedure(Sender: TObject; Port: TCommPort) - fires after a new connection was established. The Port parameter contains the serial port where the component is connected to.
OnInitialize: TNotifyEvent - fires after a connection was initalized. Connections are initialized when you open the port, if skipInit parameter is false. Otherwise you can initialize it manually, calling the InitGSMModem method.
OnDisconnect: TConnectEvent=procedure(Sender: TObject; Port: TCommPort) - fires before a disconnection. The Port parameter contains the serial port where the component was connected to.
OnNewMessage: TNewMessageEvent= procedure(Sender: TObject; Location: AnsiString; Index: Integer) - fires when new SMS message is received. Location indicates SIM memory (SM) or phone memory (ME), where the message is stored. Index is the index number of the message in the

memory location.
OnReadMessage: TReadMessageEvent = procedure(Sender: TObject; MessageText: WideString; PhoneNumber, CenterNumber: AnsiString; TimeStamp: TDateTime; TimeZone, Status, Index: Integer) - fires when an SMS is requested by GetSMS method or when new SMS is received. MessageText contains the text of the messaget, PhoneNumber is the number of the caller and CenterNumber is the number of the SMS center. TimeStamp is the date/time, when the message was received, TimeZone represents the current time zone relation to GMT, one unit is 15min. Index is the index number of the messages inside the storage location.
Status can have the following values:
-1 - Unknown status
0 - Received unread message (that is new message).
1 - Received read message.
2 - Stored unsent message.
3 - Stored sent message.
16 - Template message.
OnFault: TConnectEvent=procedure(Sender: TObject; Port: TCommPort) - occures when the serial port communication is faulted. E.g.: when you unplug an USB device which communicates on a virtual serial port.
OnError: TSendReceiveEvent=procedure(Sender: TObject; Port: TCommPort) - fires when the GSM modem reports an error in the command execution.
OnContactLoad: TContactLoadEvent=procedure(Sender: TObject; const index: Integer; const name: AnsiString; const phoneNumber: AnsiString; const location: AnsiString) - fires when a contact is loaded from the phone book.
OnTimeout: TNotifyEvent - fires when there is no data received several milliseconds (idle), the interval is specified in the ConnectionTimeout property.
OnIdle: TNotifyEvent - fires when there is no data received several milliseconds, the interval is specified in the IdleInterval property.
OnResume: TNotifyEvent - Occurs when the receiver is idle and data is received.
OnDetect: TDetectEvent=procedure(Sender: TObject; const Port: TCommPort; const BaudRate: TBaudRate; var Cancel: Boolean) - occurs when the gsm modemdetection process is in progress and there are new values of port or baud rate to be checked. If you set the Cancel parameter to true, then the detection will be cancelled.
OnReconnect: TReconnectEvent=procedure(Sender: TObject; const Port: TCommPort; const BaudRate: TBaudRate; var Cancel: Boolean) - occurs when the serial port is trying to reconnect, after fault. AutoReconnect must be true. If you set the Cancel parameter to true, then the reconnection will be cancelled.

Buy Now!

http://www.zylsoft.com
info@zylsoft.com