SL84Comm2.ocx

quick reference manual

Cardware

Contents

1 Overview

	Properties*	Methods	Events
1.	ActionArr	AboutBox	Click
2.	AsciiFileName	AddIDElement	DblClick
3.	BaudRate	DestroyTerminal	ErrMessage
4.	CardArr	EraseMessage	ModemMsg
5.	CurrDayType	FreeMutex	RxRegs
6.	CurrWeekType	GetMutex	
7.	DateArr	GetStatus	
8.	DayOfWeek	InitTerminal	
9.	DipSwitch	KillIDElement	
10.	EEPROMLoc	KillIDT	
11.	ErrCode	ModemCancelDial	
12.	FatalResetNum	ModemDial	
13.	Firmware	ModemGetStatus	
14.	IDTCurrRx	ModemRead	
15.	IDTElement	ModemWrite	
16.	IDTNum	PCTimeCopy	
17.	IDTPrevRx	ReadEEPR	
18.	InBuffer	ReconstructData	
19.	Ready	ResetCurrTx	
20.	ResertNum	RxDayType	
21.	RTC	RxIDTElement	
22.	TermMessage	RxWeekType	
23.	TermNoArr	SendMessage	
24.	TimeArr	TakeData	
25.	Total	TakeIDTNum	
26.	WorkOnLine	TxDayType	
27.		TxWeekType	
28.		WriteEEPR	
29.		WriteTime	

*In case of Visual C++, properties are accessed by SetXXX and GetXXX functions, where XXX represents an actual property name.

2 Initialization

To begin communication with SL-83 or SL-84 terminal, call the **InitTerminal** method. The control is initially off-line, so it won't gather any registrations. To put the control into the on-line state and to begin registration gathering, call the **FreeMutex** method. To put the control back to off-line state, call the **GetMutex** method.

In case of communication with the SL-850 modem module, you should call **ModemDial** after **InitTerminal**, then you should check whether connection was established in **ModemMsg** event, and finally call **FreeMutex** in case of success.

3 Properties

3.1 Registrations

Regstrations are accessed through the following properties:

Name	Type	Description
ActionArr	string array	Action codes array (see SL-83/84 technical manual)
CardArr	string array	Eight-digit identification ID media code array
DateArr	string array	Dates array as "dd.mm.yy"
TimeArr	string array	times array as "hh:mm"
TermNoArr	string array	terminal ID's array
Total	long	number of registrations received

Each array has as many elements as the number of registrations received. The number of registrations is received as a parameter in the **RxRegs** event. When processing the RxRegs event, you should form a loop based on that parameter to read all registrations. For example, the third registration is represented by the third element of each array mentioned above.

3.2 Status

To get the terminal's status data, call **GetStatus** method. Then, read the following properties:

Name	Type	Description
Firmware	String	Terminal's firmware version
DTC	String	terminal date and time

InBuffer	string	buffered registrations
DayOfWeek	string	day of week (0 - Monday, 1 - Tuesday etc...)
ResertNum	int	number of resets
FatalResetNum	int	number of fatal resets
DipSwitch	int	terminal ID

3.3 Sending messages to the terminal

Some versions of the terminal allow you to display a message on their screen. You should put your message text in **TermMessage** property, which must be exactly twenty-four characters long, and then call **SendMessage**. To erase message from terminal's screen, call **EraseMessage**.

3.4 Communication errors

The **ErrCode** property holds the last communication error code. You should read this property in **ErrMessage** event.

ErrCode	Opis
0x31	not SOH
0x32	wrong type (neither 'S' nor 'V')
0x21	not STX
0x40	time out
0x50	RAM buffer overflow
0x60	Checksum error

3.5 Registrations history file

AsciiFileName holds the registrations history file name. The default value is **reg.txt**. If you wish to change the file name, do it before you make a call to InitTerminal.

Each registration is represented by a single line of text and looks like this:

DDMMYHHmmABBBBBBBBT

Where:

DD	- day (01-31)
MM	- month (01-12)
Y	- year (1 = 2001, 2 = 2002, 3 = 2003, ...)
HH	- hour (00-23)
mm	- minute (00-59)

A	-	event code
BBBBBBBB	-	media ID
T	-	terminal code

3.6 Access tables

Name	Type	Description
CurrDayType	String	Current day type. See **RxDayType** for details.
CurrWeekType	String	Current week type. See **RxWeekType** and **RxIDTElement** for details.
IDTElement	String	Current ID table element. See **RxIDTElement** for details.
IDTNum	String	Number of ID table elements. See **TakeIDTNum**.
IDTCurrRx	Int	For internal use.
IDTPrevRx	int	For internal use.

3.7 Parameters

Name	Type	Description
WorkOnLine	Bool	This property has no effect on the control's state and vice versa. If you wish, you could set this property to True or False when calling FreeMutex or GetMutex if you want to remember your control's state.
Ready	Bool	You should use this property when processing registrations in **RxRegs** event. When set to False, it disables arrival of new registrations so that the old ones could be processed entirely. When finished, set value to True to continue receiving registrations.
BaudRate	long	Communication speed. Default iz 9600 Bd and it can be changed to 4800 Bd for long and noisy current loops
EEPROMLoc	string array	Local copy of terminal's EEPROM. For internal use only.

4 Events

4.1 ErrMessage

This event occurs in case of a communication error. When processing this event, see **ErrCode** to get error code.

4.2 RxRegs

Visual Basic
RxRegs (num As Integer)

VisualC++
void OnRxRegs(short num)

This event occurs in case of reception of a registration or a group of registrations. There's no other way to acquire registration data but to process this event. Registration data is described in chapter **3.1**.

Arguments:

num - number of registrations received and waiting to be processed.

4.3 Click

This event occurs in case of single click on a control's window.

4.4 DblClick

This event occurs in case of double click on a control's window.

4.5 ModemMsg

Visual Basic
ModemMsg (res As Integer)

VisualC++
void OnModemMsg(long res)

This event occurs after modem dialling is finished.

Arguments:

res - error code:

1 CONNECT (connected successfully)

2	RING
3	NO CARRIER
4	ERROR
6	NO DIALTONE
7	BUSY
8	NO ANSWER
10	FATAL ERROR
11	CANCELED

5 Methods

5.1 InitTerminal

Visual Basic
InitTerminal (portNum As Integer) As Boolean

Visual C++
BOOL InitTerminal (short portNum)

This method initializes the SL8xComm2 control - it sets the serial port's parameters, opens the registrations history file and starts background communication.

Arguments:

portNum - serial port number (0 - COM1, 1 - COM2,...)

Return values:

True - initialization successful
False - initialization failed; serial port busy

5.2 DestroyTerminal

Visual Basic
DestroyTerminal

Visual C++
void DestroyTerminal()

Terminates the communication and releases serial port. This method is usually called when closing your application.

5.3 PCTimeCopy

Visual Basic
PCTimeCopy ( ) As Integer

Visual C++
short PCTimeCopy ( )

Copies PC time to terminal.

Return values:

0 - Success
1 - terminal busy or not connected

5.4 GetStatus

Visual Basic
GetStatus ( ) As Integer

Visual C++
short GetStatus ( )

Call this method and read properties described in chapter **1.3**.

Return values:

0 - Success
1 - terminal busy or not connected

5.5 SendMessage

Visual Basic
SendMessage (cmd As Integer) As Integer

Visual C++
short SendMessage (short cmd)

This method sets the terminal's screen message. The message stays until it is deleted by **EraseMessage** method.

Newer versions of SL-83 and SL-84 do not support this (nothing is displayed).

Assign your message (24 characters exactly) to **TermMessage** property before you call this method.

Arguments:

cmd : 71 (it's 'G', which stands for 'Global Message')

Return values:

0 - Success
1 - terminal busy or not connected

5.6 EraseMessage

Visual Basic
EraseMessage ( ) As Integer

Visual C++
short EraseMessage ( )

Erases message from terminal's screen.

Return values:

0 - Success
1 - terminal busy or not connected

5.7 TakeData

Visual Basic
TakeData ( ) As Integer

Visual C++
short TakeData ( )

This method instructs the terminal to send a block of registrations immediatelly. A block of registrations consists of 32 registrations (or less, if buffer holds less than 32 registrations). You can acquire these registrations in **RxRegs** event.

Return values:

0 - Success
1 - terminal busy or not connected

5.8 ReconstructData

Visual Basic
ReconstructData ( ) As Integer

Visual C++
short ReconstructData ( )

This method instructs the terminal to reconstruct (pack) registration data in its memory. This is a time consuming process and you should call this method only in case of some major data loss on your acquisition computer.

Return values:

0 - Success
1 - terminal busy or not connected

5.9 AddIDElement

Visual Basic
AddIDElement (ibutton As String, abyte As Integer) As Integer

Visual C++
short AddIDElement (LPCTSTR ibutton, short abyte)

This method adds a new element into the terminal's ID table. An element consists of a:

[ibutton, week type]

pair.

Arguments:

ibutton – ID media number (without crc byte, see SL-83 and SL-84 tech. docs)
abyte – week type (0 to 31) and action bits (see SL-83 and SL-84 tech. docs)

Return values:

0 - OK
1 - terminal busy or not connected
-2 - transfer error
-3 - table full, element not written
-4 - missing control character 0 or 1 after ACK (see ID table writing protocol)
-10 - missing control character ACK after an eleme nt's reception (see ID table writing protocol)
-11 - neither ACK nor NAK after an element's reception (see ID table writing protocol)

For example, a touch memory button ID is engraved at the button's surface and looks something like this:

```
60        01
0000017A67A3
```

In this example, the entire touch memory code is 600000017A67A301 (16 hexadecimal digits, 8 bytes), and the CRC byte is 60H (upper left corner). You should provide the following string as the ibutton argument: 0000017A67A301.

5.10 RxIDTElement

Visual Basic
RxIDTElement ( ) As Integer

Visual C++
short RxIDTElement ( )

Call this method to read ID table elements. You should read the table entirely by calling this method repeatedly until it returns 4 (end of ID table). Each call to this method sets values of **CurrWeekType** and **IDTElement** properties to those of the currently read element.

Return values:

0	- OK
2	- terminal busy or not connected
4	- end of ID table, no more elements
96	- communication error
100	- the same element sent twice

5.11 KillIDElement

Visual Basic
KillIDElement (ibutton As String, abyte As Integer) As Integer

Visual C++
short KillIDElement(LPCTSTR ibutton, short abyte)

This method erases an ID element ([touch memory, week type] pair),

Arguments:

ibutton – ID media number (without crc byte, see SL-83 and SL-84 tech. docs)
abyte – week type (0 to 31) and action bits (see SL-83 and SL-84 tech. docs)

Return values:

0	- OK
1	- busy or not connected
-2	- transfer error
-10	- missing control character ACK after an element's reception (see ID table deleting protocol)
11	- neither ACK nor NAK after an element's reception

5.12 KillIDT

Visual Basic
KillIDT() As Integer

Visual C++
short KillIDT ( )

Erases entire ID table.

Return values:

0 - Success
1 - terminal busy or not connected

5.13 TakeIDTNum

Visual Basic
TakeIDTNum ( ) As Integer

Visual C++
short TakeIDTNum ( )

This method sets **IDTNum** property to number of ID table elements.

Return values:

0 - Success
1 - terminal busy or not connected
-10 - communication error

5.14 RxDayType

Visual Basic
RxDayType ( ) As Integer

Visual C++
short RxDayType ( )

Use this method to read the terminal's day-type table elements (32). You should read the entire table, calling the method within the loop, until you reach the end of table (that is when the method returns the value 4). The method returns the error code, while the currently read element is stored in the **CurrDayType** property, in the following form:

<n><FH1><FM1><TH1><TM1><FH2><FM2><TH2><TM2>

<n> - day type ID (0 to 31)

<FH1> - first access interval, from hour
<FM1> - first access interval, from minute
<TH1> - first access interval, to hour
<TM1> - first access interval, to minute
<FH2> - second access interval, from hour
<FM2> - second access interval, from minute
<TH2> - second access interval, to hour
<TM2> - second access interval, to minute

All numbered elements of the **CurrDayType** property are ASCII characters (number 0 (min. value) is represented as ASCII character '0' (30H); the number 59 (max. value) is ASCII character 'k'(6BH)).

For example, day type 16 whose access intervals are defined as:

From 08:00 to 12:00 and from 20:30 to 23:59

the terminal sends in the following form

@80<0DNGk

@ = 40H (40H – 30H = 10H = 16 dec.), 8 = 38H, 0 = 30H etc...

Return values:

0 - OK
4 - end of table
1 - terminal busy or not connected
96 - communication error (control bytes don't match)
100 - the same element sent twice
-100 - communication error (garbage)
>250 - transfer aborted

5.15 TxDayType

Visual Basic
TxDayType(day As String) As Integer

Visual C++
short TxDayType (LPCTSTR day)

Use this method to write the day types to the terminal's day type table. You should always download the entire table to the terminal (call the method within the loop for 32 times).

Arguments:

day – day type definition. It must have the following form:

<n><FH1><FM1><TH1><TM1><FH2><FM2><TH2><TM2>

<n> - day-type ID
<FH1> - first access interval, from hour
<FM1> - first access interval, from minute
<TH1> - first access interval, to hour
<TM1> - first access interval, to minute
<FH2> - second access interval, from hour
<FM2> - second access interval, from minute
<TH2> - second access interval, to hour
<TM2> - second access interval, to minute

All numbered elements of the **CurrDayType** property are ASCII characters (number 0 is represented as ASCII character '0' (30H), number 59 as ASCII character 'k' (6BH; 6BH – 30H = 3BH = 3*16 + 11 = 59 dec.)).

For example, day type 16 whose access intervals are defined as:

From 08:00 to 12:00 and from 20:30 to 23:59

you should pass as an argument to this method in the form:

@80<0DNGk

@ = 40H (40H – 30H = 10H (16 dec.)), 8 = 38H (38H – 30H = 8H (8dec)), 0 = 30H etc...

5.16 RxWeekType

Visual Basic
RxWeekType ( ) As Integer

Visual C++
short RxWeekType ( )

Use this method to read the terminal's day type table elements (32). You should read the entire table, calling the method within the loop, until you reach the end of table (that is when the method returns the value 4). The method returns the error code, while the currently read element is stored in the **CurrWeekType** property, in the following form:

<ID><Sunday><Monday><Tuesday><Wednesday><Thursday><Friday><Saturday>

All numbered elements of the **CurrWeekType** property are ASCII characters (number 0 (min. value) is represented as ASCII character '0' (30H); the number 32 (max. value) is ASCII character 'P' (50H)).

For example, the week-type 16, which is defined as:

Day of the week	day-type
Sunday	32
Monday	01
Tuesday	01
Wednesday	02
Thursday	15
Friday	19
Saturday	22

the terminal sends in the following form:

@P112?CF

@ = 40H (40H – 30H = 10h = 16dec), P = 50H (50H – 30H = 20H = 32dec), etc...

5.17 TxWeekType

Visual Basic
TxWeekType (week As String) As Integer

Visual C++
short TxWeekType (LPCTSTR week)

Use this method to write the week-types to the terminal's week-type table. You always should download the entire table to the terminal (call the method within the loop for 32 times. See the examples).

Arguments:

week – week-type definition, which must be in the form:

<ID><Sunday><Monday><Tuesday><Wednesday><Thursday><Friday><Saturday>0

All numbered elements of the **CurrWeekType** property are ASCII characters (number 0 (min. value) is represented as ASCII character '0' (30H); the number 32 (max. value) is ASCII character 'P' (50H)).

For example, the week-type 16, which is defined as:

Day of the week	day-type
Sunday	32
Monday	01
Tuesday	01
Wednesday	02
Thursday	15
Friday	19
Saturday	22

you should pass as an argument to the **TxWeekType** method as the string value:

@P112?CF0

@ = 40H (40H – 30H = 10h = 16dec), P = 50H (50H – 30H = 20H = 32dec), etc...

At the end of the week-type definition string always add the character 0 (required by the protocol!)

5.18 ReadEEPR

Visual Basic
ReadEEPR (address As Integer) As Integer

Visual C++
short ReadEEPR(short address)

Reads EEPROM value from given address.

5.19 WriteEEPR

Visual Basic
WriteEEPR (address As Integer, contents As Long) As Integer

Visual C++
short WriteEEPR(short address, long contents)

Writes value to EEPROM's given address.

5.20 AboutBox

Visual Basic
AboutBox

Visual C++
void AboutBox()

Displays information about the control and authors.

5.21 FreeMutex

Visual Basic
FreeMutex

Visual C++
void FreeMutex()

Puts control in on-line state nad begins registrations gathering.

5.22 GetMutex

Visual Basic
GetMutex

Visual C++
void GetMutex()

Puts control in off-line state nad stops registrations gathering.

5.23 WriteTime

Visual Basic
WriteTime (th As Integer, tm As Integer, dd As Integer, dm As Integer, dy As Integer, dow As Integer) as Integer

Visual C++
short WriteTime(short th, short tm, short dd, short dm, short dy, short dow)

Sets terminal's date and time.

Arguments:

th - hour
tm - minute
dd - day
dm - monts
dy - year
dow - day of week (0 - Monday, 1 - Tuesday,...)

5.24 ResetCurrTx

Visual Basic
ResetCurrTx () As Integer

Visual C++
short ResetCurrTx()

Resets terminal's internal ID table, day-types and week-types table pointer. Please note that the same pointer is internaly used for all three types of tables, which makes it possible only to read one table at the time.

5.25 ModemDial

Visual Basic
ModemDial (telNum As String) As Integer

Visual C++
long ModemDial (LPCTSTR telNum)

Makes a phone call to remote SL-850 modem module and fires **ModemMsg** event. See ModemMsg event for error codes.

Arguments:

telNum - phone number, including modem initialization string.

5.26 ModemCancelDial

Visual Basic
ModemCancelDial ()

Visual C++
void ModemCancelDial ()

Cancels dialing.

5.27 ModemGetStatus

Visual Basic
ModemGetStatus () As Integer

Visual C++
long ModemGetStatus ()

Returns modem status. See API function **GetCommModemStatus** for details. You should use this method to check whether modem is on-line.

5.28 ModemRead

Visual Basic
ModemRead() as String

Visual C++
BSTR ModemRead()

Reads a string from an opened serial port.

Return values:
- string read from the device

5.29 ModemWrite

Visual Basic
ModemWrite(data As String) as Integer

Visual C++
long ModemWrite(LPCTSTR data)

Writes a string to an opened serial port.

Arguments:

data - string to write

Return values:

- number of characters written