SCardX Easy

Smart Card ActiveX control
Version 1.3

Smart Cards in the Borland C++ Builder applications

Developers Manual

Document ver.1.2
Dec. 22, 2005

Smart Cards in the Borland C++ Builder applications. Developers Manual.

Printed: Dec. 22, 2005

Publisher

SCardSOFT

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

Thank You for your interest to the SCardX Easy smart card ActiveX control!

Please send me all your suggestions or any questions about the SCardX Easy smart card ActiveX control via e-mail igor@scardsoft.com.

Visit our web site for the latest software and specifications updates.

Yours,
Igor V. Kharchenko
author.

Table of Contents

1 About

1.1 About SCardX Easy ActiveX control

SCardX Easy

Smart Card ActiveX control

Version 1.3

1.2 Contacts

The official web site of SCardX Easy is the SCardSOFT homepage:

useful SCardSOFT pages:

SCardX Easy official web page

SCardSOFT Home

Smart Cards specifications Library page

Smart Cards Forum (English)
Smart Cards Forum (Russian)

Prices page
License's purchasing info page

contact e-mail addresses:

info@scardsoft.com - common questions;
sales@scardsoft.com - payments and licenses questions;
support@scardsoft.com - support service;

2 SCardX Easy ActiveX control overview

2.1 What is the SCardX Easy?

2.1.1 SCardX Easy is an ActiveX

SCardX Easy is a standart ActiveX control.

If your development environment (IDE) supports the ActiveX technology like the MS Visual Studio, Borland Delphi or C++ Builder or other - than the SCardX Easy may be successfully used by your applications.

2.1.2 What SCardX Easy can to add into your application?

SCardX Easy adds to your applications the following functionality:

smart cards functionality :

- receiving the smart card service's and devices' events;
- receiving an information about the attached devices;
- receiving an information about the opened smart card;
- sending the command data buffers into the opened smart cards and receiving the cards responses;
- managing the cards opening and closing modes;

additional useful tools :

- Error LookUp and Reader States LookUp services
- Data ciphering
- Tray Icon usage

2.1.3 Smart cards in your applications

The SCardX Easy ActiveX control creates the communication channel between the parent application and an opened smart card via the smart card service and any attached PC/SC compatible smart card reader.

The SCardX Easy allows you to send the command data buffers into any ISO-7816 compatible smart cards and to receive the cards' answers.

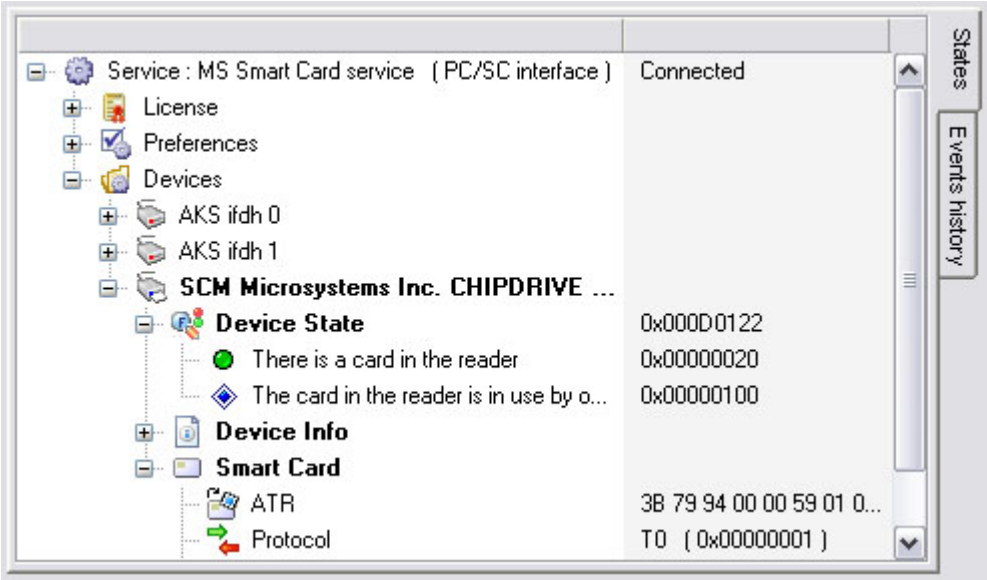
Using SCardX Easy ActiveX control you can talk with a smart card using card's "native" language - the language of the command APDU's. It is the lowest level of work with a smart cards from the PC.

Using SCardX Easy ActiveX control you can send into your cards any commands according to the cards' specifications easy and without any limitations.

2.2 Appearance

2.2.1 States page

The "States" page is a main user interface element of the SCardX Easy ActiveX control.



There are many useful information and context pop-up menu commands on this page:

Smart card service info:

- selected smart card service
- service connection state

Your License info:

- License owner's name and address
- License number
- License type
- License usage rules

Preferences:

PC/SC Card detecting defaults

- Open the reader automatically : Yes, No
- Preferred Protocol: T0, T1, RAW, Autodetect, Undefined
- Preferred Sharing Mode: Share reader, Exclusive use, Direct reader control
- Card closing mode: Live card, Reset card, Unpower card, Eject card

Miscellaneous

- Separate received HEX bytes : Yes, No
- Events logging : Log all events, Log most useful events only

Attached devices' list:

- Device state
- Device info

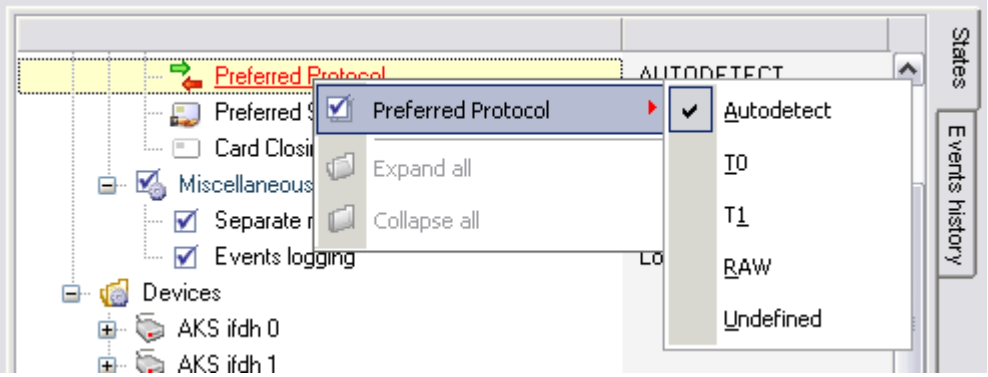
Opened smart card info:

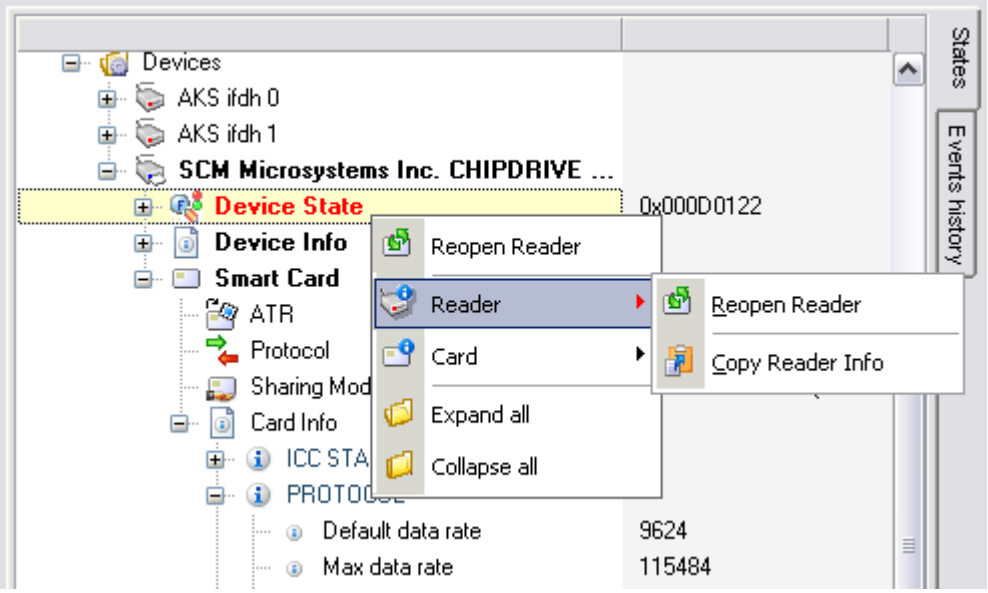
- ATR
- Protocol
- Sharing mode
- Card info

Error

- The last error info

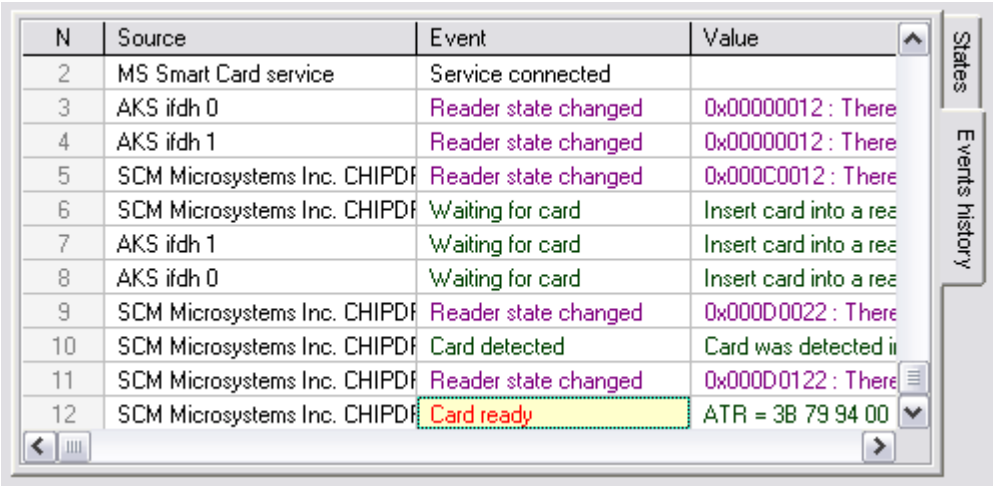
This page has the context pop-up menu which allows you to take access to many useful commands depending to the selected item.





2.2.2 Events History page

This page contents the archive of the events which was occured.



N	Source	Event	Value
2	MS Smart Card service	Service connected	
3	AKS ifdh 0	Reader state changed	0x00000012 : There
4	AKS ifdh 1	Reader state changed	0x00000012 : There
5	SCM Microsystems Inc. CHIPDF	Reader state changed	0x000C0012 : There
6	SCM Microsystems Inc. CHIPDF	Waiting for card	Insert card into a rea
7	AKS ifdh 1	Waiting for card	Insert card into a rea
8	AKS ifdh 0	Waiting for card	Insert card into a rea
9	SCM Microsystems Inc. CHIPDF	Reader state changed	0x000D0022 : There
10	SCM Microsystems Inc. CHIPDF	Card detected	Card was detected i
11	SCM Microsystems Inc. CHIPDF	Reader state changed	0x000D0122 : There
12	SCM Microsystems Inc. CHIPDF	Card ready	ATR = 3B 79 94 00

Fields

N - the serial number of the event;
Source - event source;
Event - event message;
Value - event value (if present);
Event Time - the time when the event was occured;

Pop-up Menu Commands

First Event - go to a first record;
Last Event - go to a last record;
Events logging - the logging mode : Log all events, Log most useful events only
Save Events History - save grid data to a text file;
Copy Events History - copy grid data to a Windows Clipboard;
Clear All - clear all events messages at once;

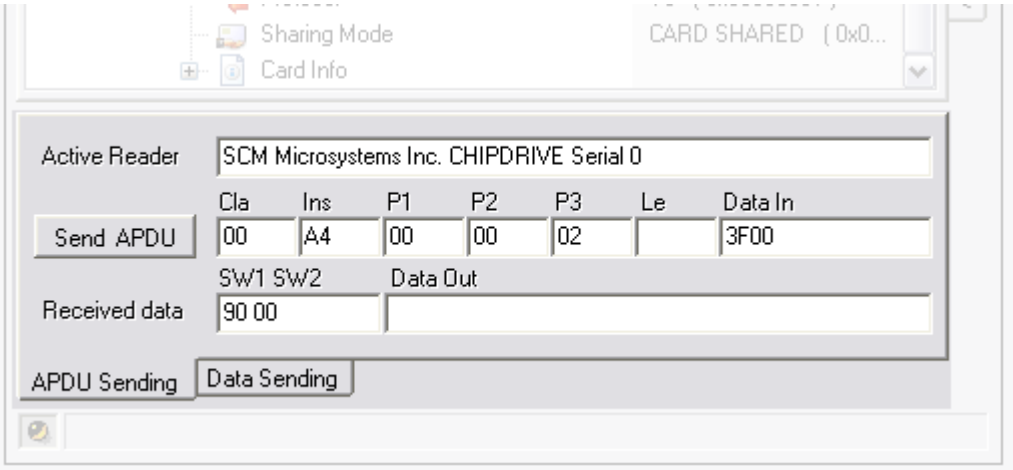
Useful info:

- you can hide/show this page by operating of the VisibleEventsHistory property;
- you can read the Events History grid data to your application by calling the function GetEventsHistory;
- you can clear the Events History grid data by calling the function EventsHistoryClear;
- you can lock/unlock the events logging by operating of the EventsHistoryEnabled property.

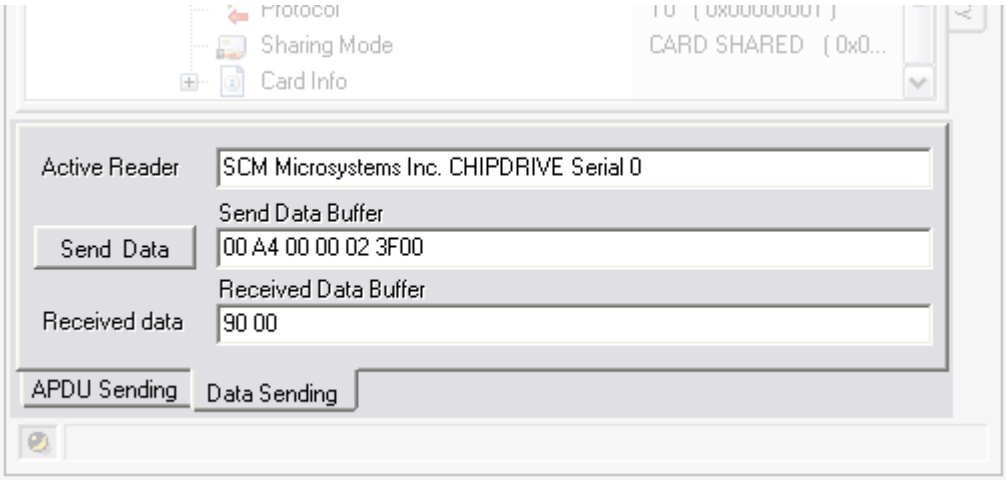
2.2.3 ToolBar panel

The ToolBar panel contain the controls for the data sending.

Using the ToolBar you can prepare and send into an opened smart card the control APDU's.



Or you can prepare and send into an opened smart card the unformatted data buffers.



The ToolBar may be used for testing of the smart card service connection or your device from any temporary application because it is ready for data sending at once after adding the SCardX Easy to your application.

If you don't need the ToolBar into your main application you can hide it easy.

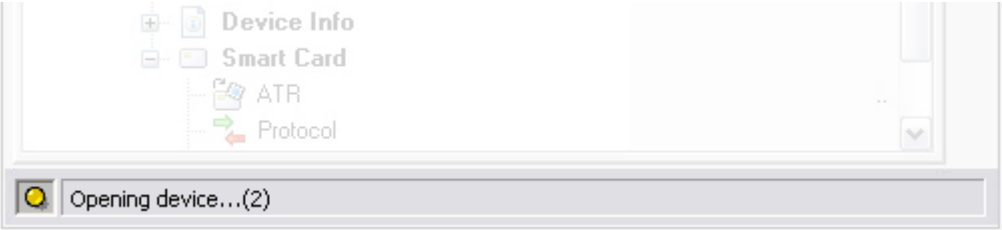
Useful info:

- you can hide/show the ToolBar by operating of the VisibleToolBar property;

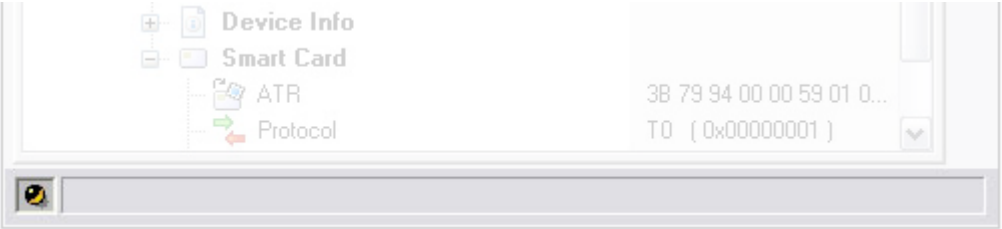
2.2.4 StatusBar panel

The StatusBar is an indicator of the activity of the data exchange process between the SCardX Easy and a smart card service.

If the control is locked the Led is On.



When the control is not locked the Led is Off.



Useful info:

- you can hide/show the StatusBar by operating of the VisibleStatusBar property;

2.3 Smart card functionality

2.3.1 Smart card service

The smart card service is a drivers' layer which is used by SCardX Easy for communication with a smart card.

Each card readers' manufacturer supports its devices by its own drivers' set.

However the last versions of the Microsoft Windows OS supports its own smart card service based on the PC/SC standard. The Microsoft PC/SC smart card service allows to any applications to work with smart cards independent to the hardware drivers.

Today SCardX Easy supports the MS Smart Card Service (PC/SC Interface) and it works with any of PC/SC compatible smart card readers.

The next versions of SCardX Easy will additionally support some another alternative smart card services .

Useful info:

- you can select the smart card service by operating of the SmartCardService property;
- you can connect SCardX Easy to the selected service or disconnect it by operating of the ConnectionState property;

2.3.2 Events

The SCardX Easy allows to your application to receive all possible events from the selected smart card service:

User interface events

OnHistoryEvent
OnReaderSelected
OnTrayIconDblClick
OnTrayIconMenuItem

Smart card work events

OnCardDetected
OnCardInvalid
OnCardReady
OnCardWait
OnConnected
OnDataSent
OnDisconnected
OnReadersList
OnReaderStateChanged

Other events

OnERROR
OnLock
OnUnlock

2.3.3 Data sending

The SCardX Easy allows to your application to send the data into a card and to receive the card answers.

The data sending functions are:

- SendCardAPDU : sending the command APDU's;
- SendCardDATA : sending the unformatted data buffers;

Before the data sending your application must prepare the sending data in the hexadecimal format according to the specification of your card.

After calling both these functions returns the hexadecimal data buffer of the card answer on the sent data.

You may analyze the card answers according to the cards' specifications.

2.4 Additional tools

2.4.1 LookUp service

The SCardX Easy allows to your application to use the following LookUp services:

- Error LookUp : decodes any error code from it number value to the text string;
- State LookUp : decodes and unpacks the readers' state code from it number value to the text string;

2.4.2 Data ciphering

The SCardX Easy allows to your application to encode and to decode the text strings using the DES algorithm.

The DES ciphering functions are:

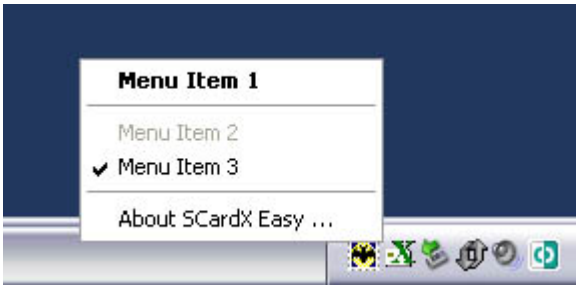
- DES_EncryptString : for encrypting the text;
- DES_DecryptString : for decrypting text from an encrypted hex data buffer;

2.4.3 Tray Icon usage

The SCardX Easy has its own icon in the system tray zone.

By default this icon has a single pop-up menu item "About...".

You can expand this pop-up menu by adding of your own menu items at any time.



The SCardX Easy allows you to add any counts of your own menu items.

Useful info:

- you can re-create the TrayIcon's menu by calling the TrayIconMenuCreate function;
- you can clear all menu items of the TrayIcon at once by calling the TrayIconMenuClear function;
- you can check/uncheck the menu item by calling the TrayIconMenuItemSetChecked function;
- you can enable/disable the menu item by calling the TrayIconMenuItemSetEnabled function;
- you can make the menu item as a default item by calling the TrayIconMenuItemSetDefault function;
- when the user clicks on the TrayIcon menu item the event OnTrayIconMenuItem occurs;
- when the user twice clicks on the TrayIcon the event OnTrayIconDblClick occurs;

2.4.4 Preferences

The SCardX Easy allows you to change the preferences via its ActiveX interface.

PC/SC Card detecting defaults

Using the SetPref_PCSC_OnCardDetect function you can set up of the following preferences:

- Open the reader automatically
- Preferred Protocol
- Preferred Sharing Mode
- Card closing mode

Miscellaneous

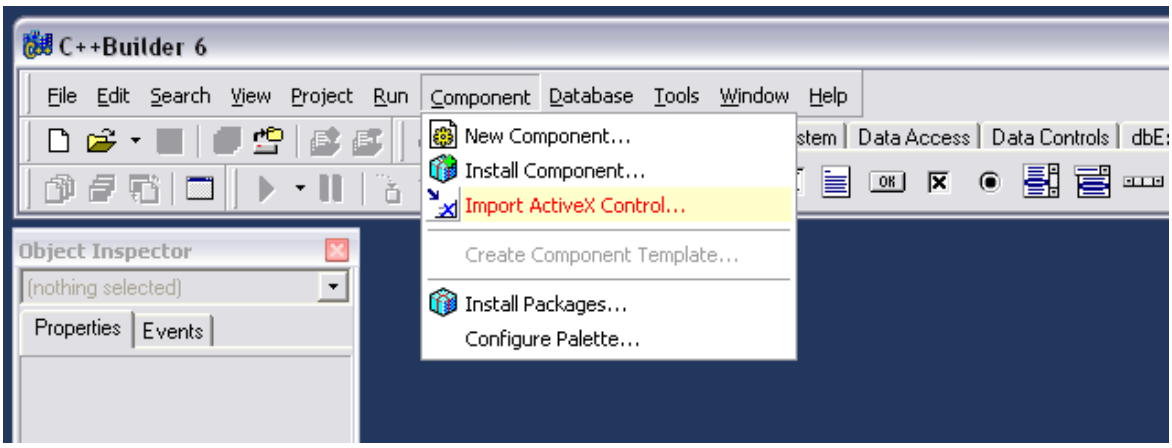
Using the SeparateReceivedBytes property you can set up the "Separate received HEX bytes" parameter of the control's preferences.

Using the EventsLogging property you can set up the "Events logging" parameter of the control's preferences.

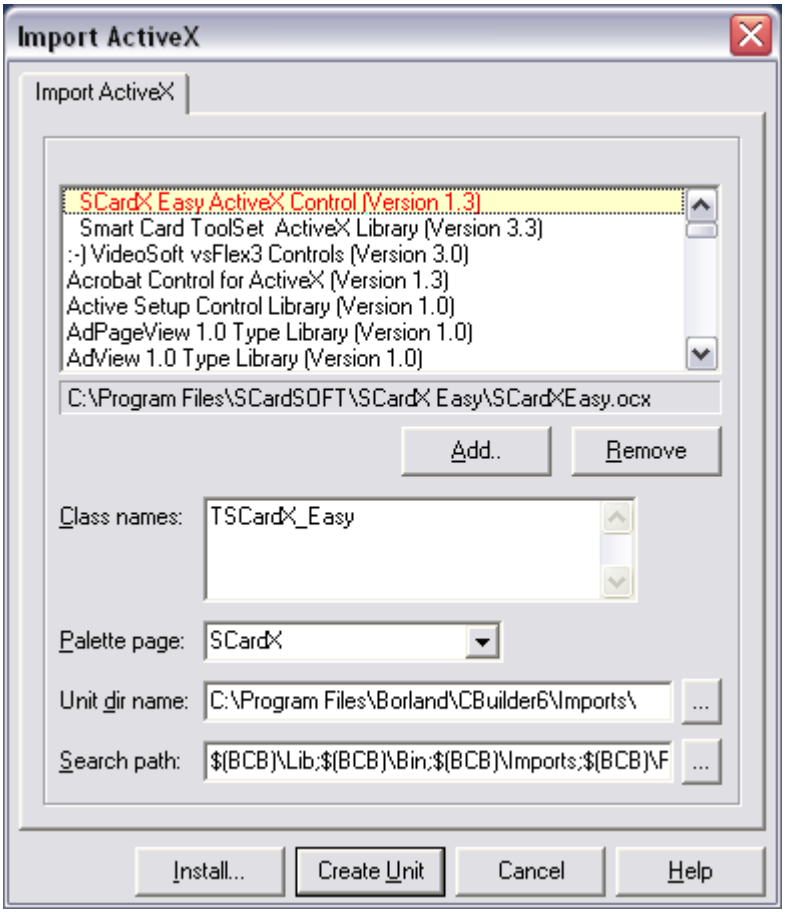
3 SCardX Easy first start

3.1 Registering SCardX Easy ActiveX control on the Borland C++ Builder 6 IDE components palette

Open the Borland C++ Builder 6 IDE main menu Component and click on the item "Import ActiveX Control..."



Select the "SCardX Easy ActiveX Control" string in the controls' list of the window "Import ActiveX"

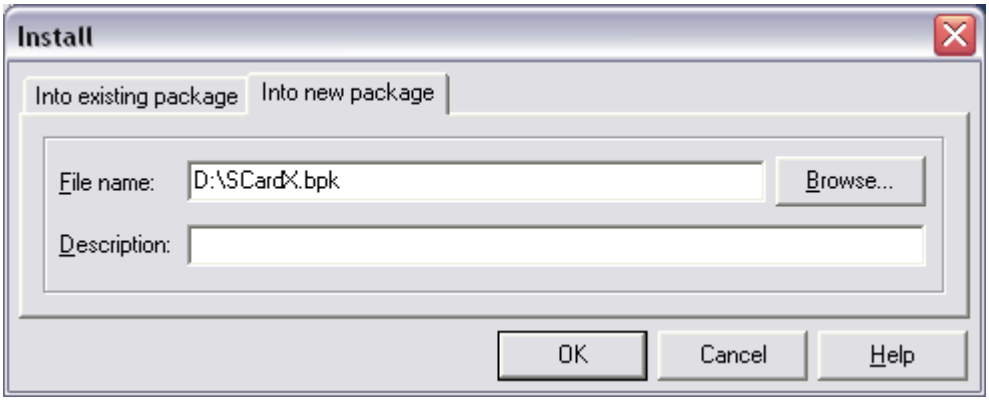


Please do not change the class name "TSCardX_Easy" because this name is used by our C++ Builder demo application.

Define the name of the component palette "Palette Page" where C++ Builder will place the component icon.

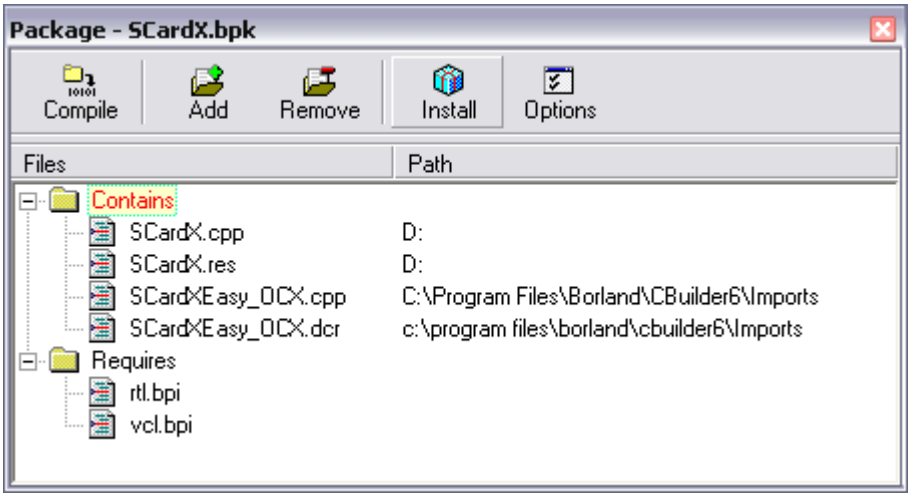
Click on the "Install" button.

The window "Import ActiveX" will be closed and the window "Install" will be opened.



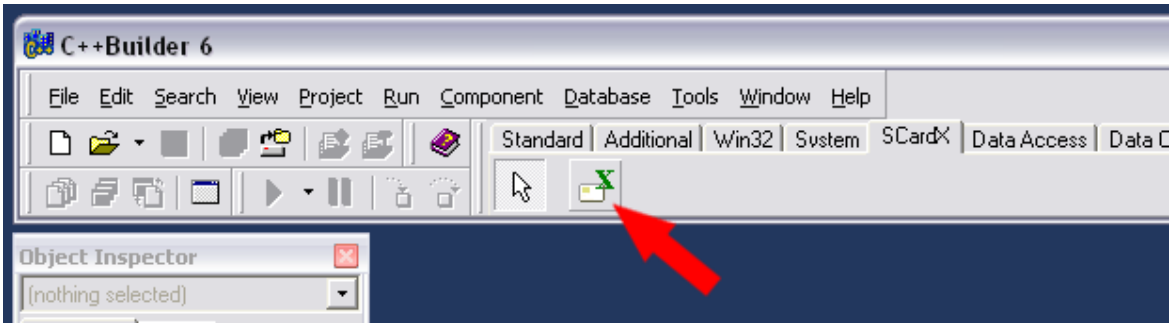
Define the components package where C++ Builder will insert a new component and click on the "Ok" button.

The window "Install" will be closed and the window "Package" will be opened.



Click on the "Install" button.

The window "Package" will be closed and the new ActiveX component will be created on the C++ Builder components palette.

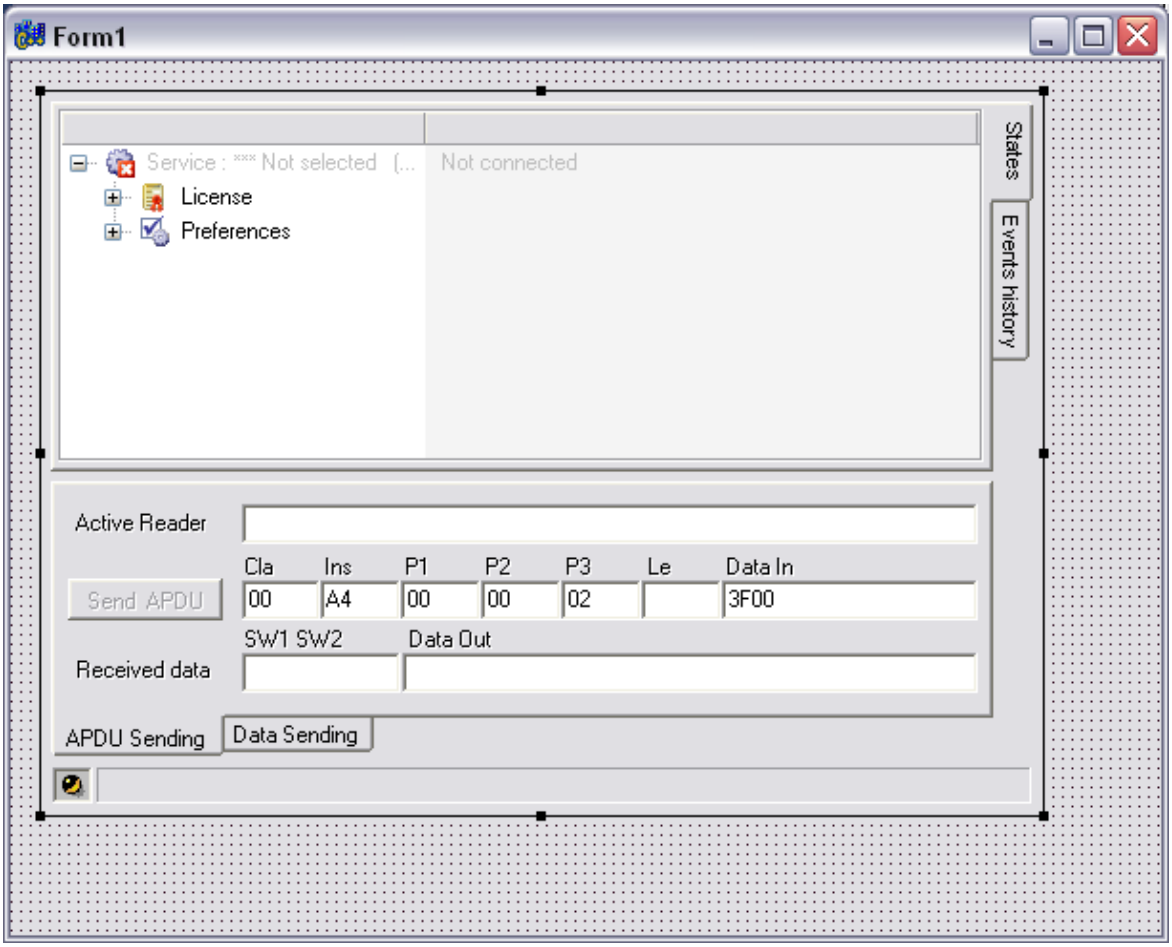


You can use now the SCardX Easy ActiveX control as a C++ Builder component in your applications.

3.2 Your first application and the connection testing

You can create the first small application for testing of the smart card service and card readers of your PC .

Please create the new C++ Builder application. Drag the SCardX_Easy icon from the components palette and drop it on the new form.



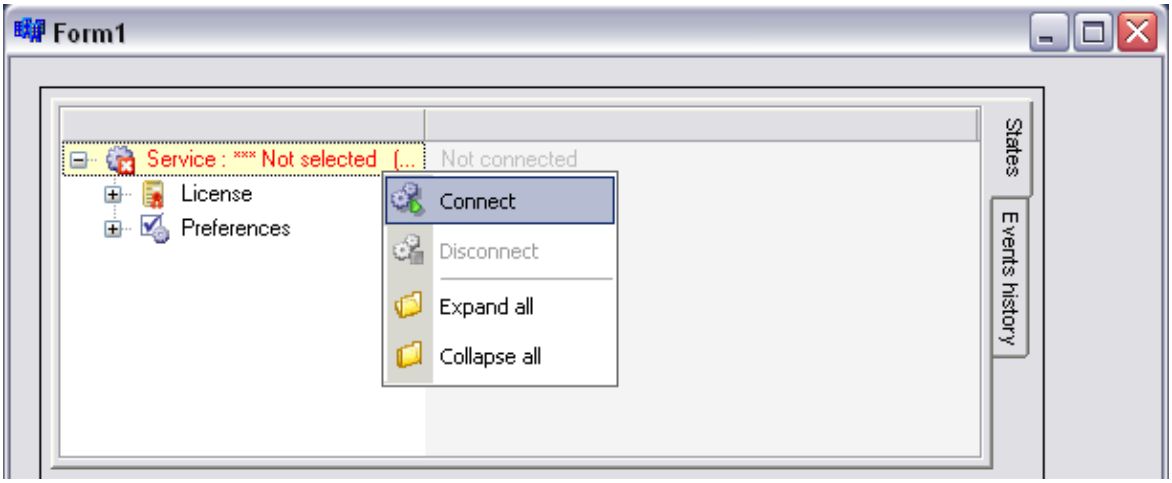
Set up the VisibleToolBar property of the SCardX Easy to true.

Go to the Form1.OnClose event and place the following command there:

```
void __fastcall TForm1::FormClose(TObject *Sender, TCloseAction &Action)
{
SCardX_Easy1->Finalize();
}
```

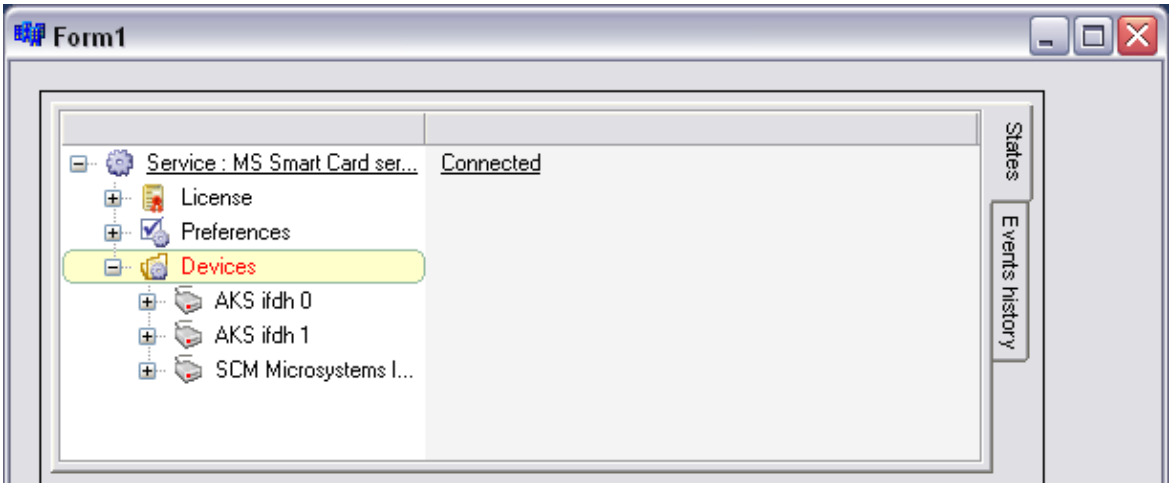
Run it.

Click on the "Service" item of the "States" page of the SCardX Easy by the right mouse button and select the menu item "Connect ":



The SCardX Easy ActiveX control will try to connect the MS Smart Card service.

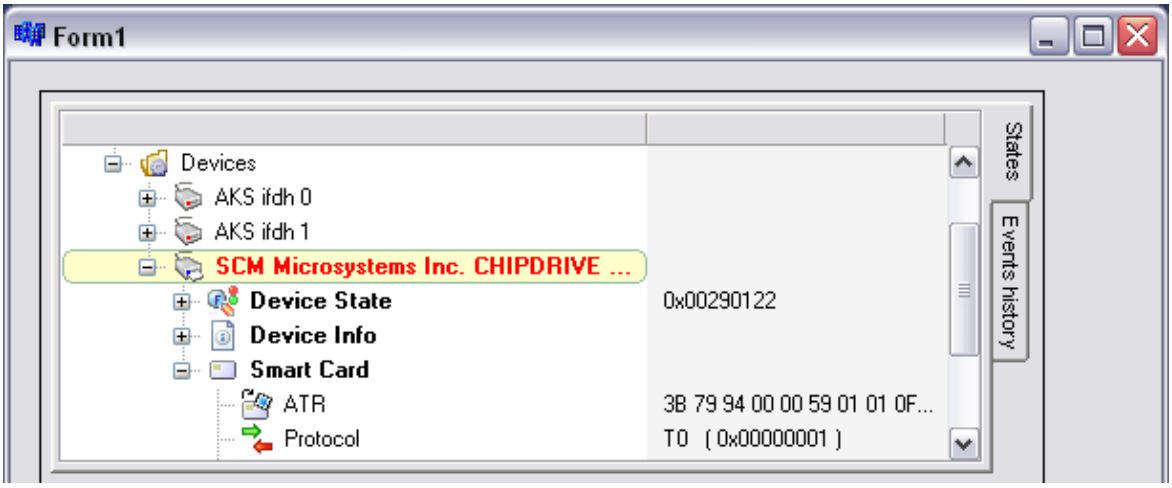
If these drivers are present on your PC the SCardX Easy ActiveX control will connect its and the available card readers names list will be shown.



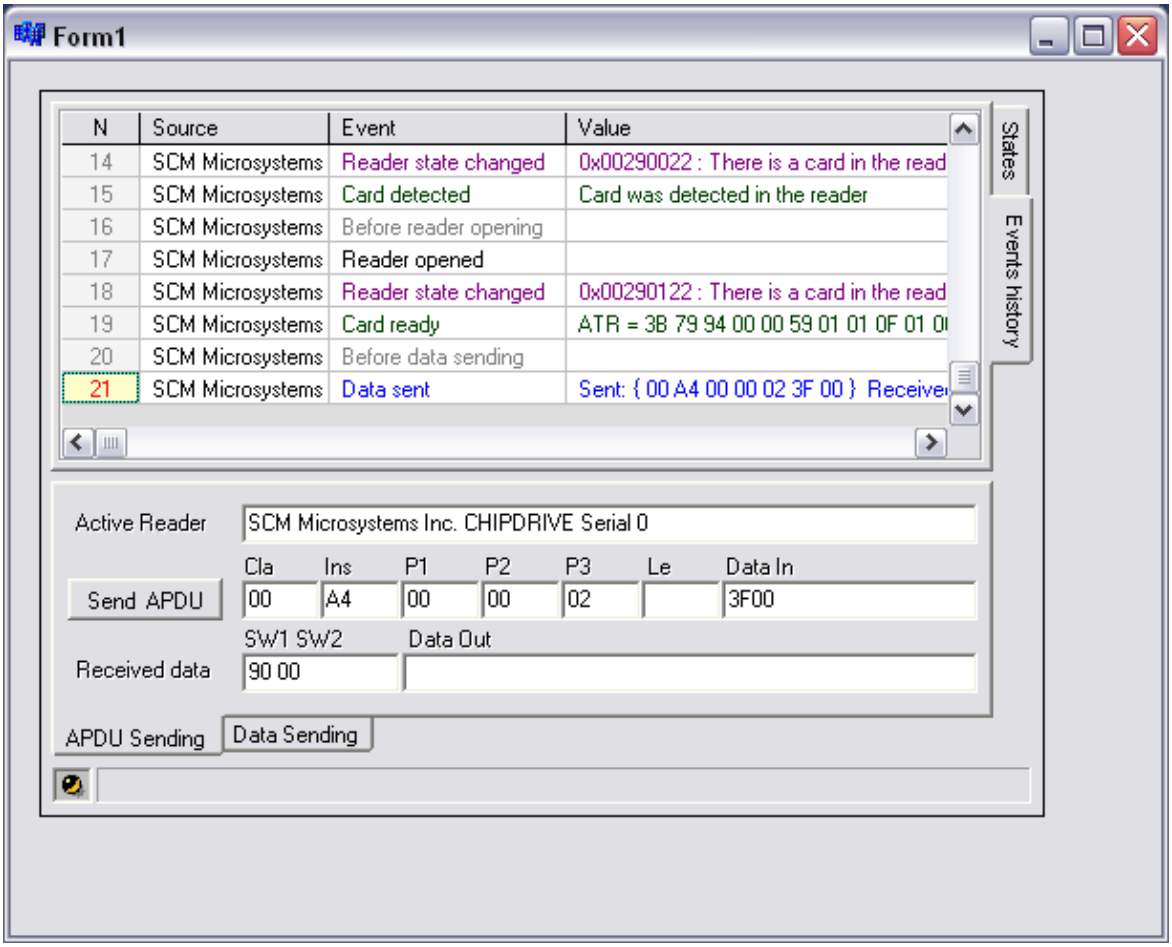
Insert the standart ISO-7816 smart card like the GSM SIM into the reader.

Warning! Do not use any memory cards for this test!

If the card is valid it will be opened and the info about of this card will be shown on the "States" page.



Click on the highlighted reader item.
Open the "Events History" page.
Click on the "Send APDU " button of the ToolBar panel.



If the data will be sent into the card correctly:

- the event "Data sent" will be occured and placed into the events history grid;
- the received card answer will be placed into the "Received data" controls of the ToolBar panel;

Otherwise an error event will be created and placed into the events history grid.

That's all.

If you can send now the data buffers into your cards you may start to create your first smart cards application.

If an error event will be occured during of this test it means that either the smart card service on the your PC is not started or your devices are not works. In this case you can contact the SCard SOFT's support service via e-mail support@scardsoft.com for detecting and removing the troubles.

4 Your first application. " Hello, cards World ! "

4.1 Demo application

The SCardX Easy setup program installs the source codes of example applications.

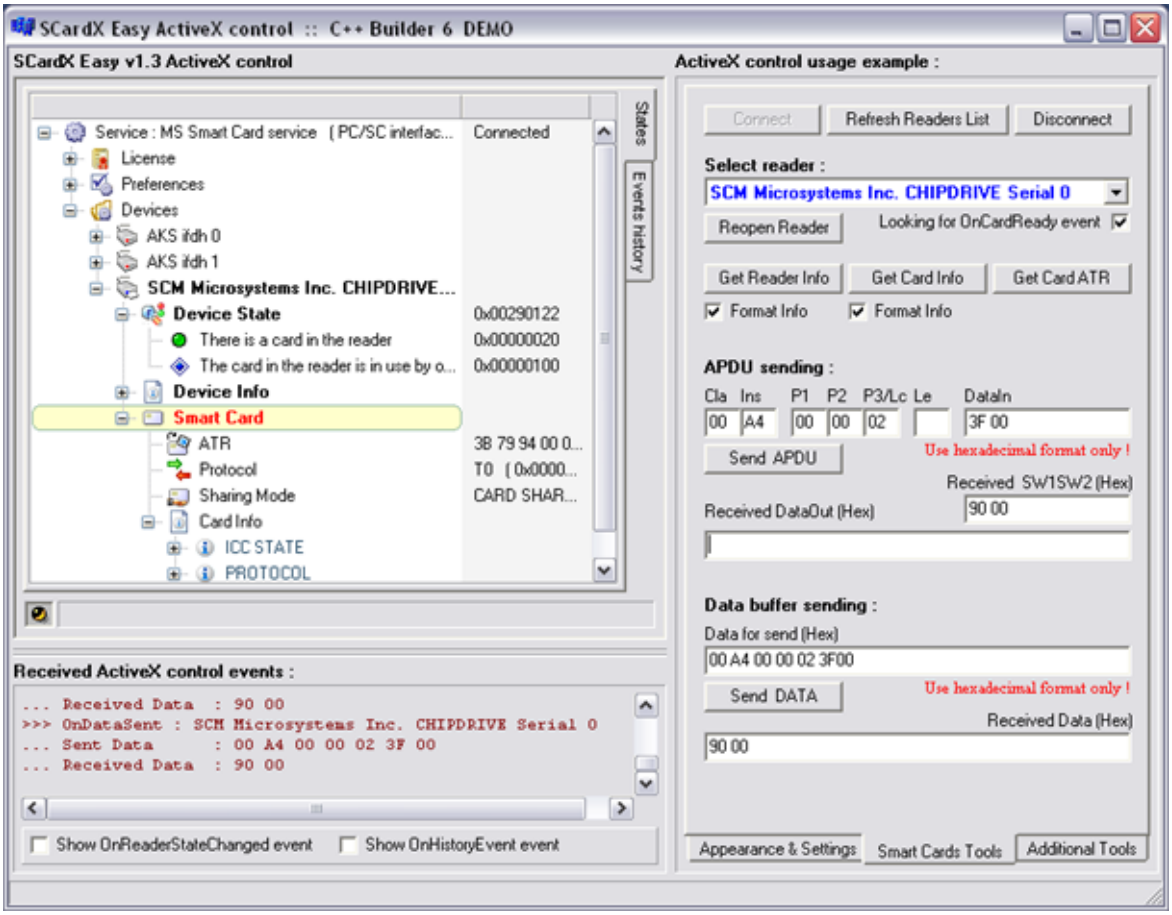
The default examples' path on your hard drive after control's installation is:

```
"C:\Program Files\SCardSOFT\SCardX Easy\- Examples"
```

You can find the Borland C++ Builder 6 demo application source codes on the following default path:

```
"C:\Program Files\SCardSOFT\SCardX Easy\- Examples\Borland C++ Builder 6"
```

The compiled demo application looks like on this picture:

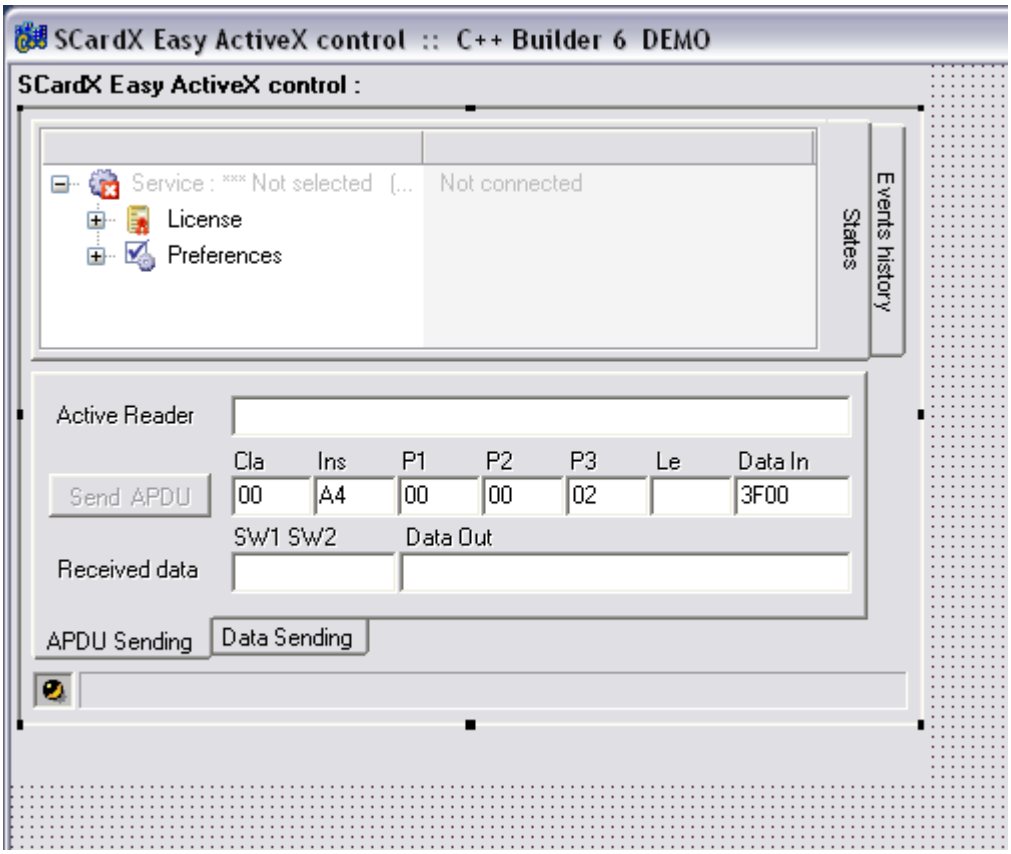


This demo application will be used by this Manual as a base of your first smart cards C++ Builder application.

Please find it and copy its source codes to your C++ Builder projects workplace.

4.2 New C++ Builder project

1. Create the new C++ Builder project.
2. Rename the form from Form1 to MainForm.
3. Drag the SCardX_Easy icon from the components palette and drop it on the new form.
4. Rename the SCardX Easy from SCardX_Easy1 to SCardX_Easy.
5. Set up the control's position on the form.



4.3 Interface functions

You need to control the states of the form's controls depending to the states of the connection and to your readers' states.

For example the data sending command button must be disabled while the reader is empty.

For managing of the controls' states you need to control the values of the following three SCardX Easy ActiveX control properties:

ConnectionState
IsLocked
IsCardReady

When one of these properties becomes changed you need enable or disable some of the form's controls.

The demo application has two interface functions:

```
// enable/disable the controls of the connection oriented commands
void __fastcall TMainForm::EnableControls()
{
  try
  {
  WideString wReaderName      = ReadersList->Text;
  bool bb                     = (SCardX_Easy->ConnectionState == stServiceConnected);
  Button2->Enabled            = (! bb);
  Button3->Enabled            = bb ;
  Button4->Enabled            = bb ;
  ReadersList->Enabled        = bb ;
  }
  __finally
  {
  EnableCardControls();
  }
}
```

```
// enable/disable the controls of the smart card commands
void __fastcall TMainForm::EnableCardControls()
{
  try
  {
  WideString wReaderName = ReadersList->Text;
  bool bb                = SCardX_Easy->IsCardReady(&wReaderName);

  Button5->Enabled  = bb;
  Button6->Enabled  = bb;
  Button7->Enabled  = bb;
  Button8->Enabled  = bb;
  Button9->Enabled  = bb;
  Button10->Enabled = bb;
  }
  __finally
  {
  }
}
```

Call the function EnableControls on the following events:

OnConnected
OnDisconnected
OnLock
OnUnlock

The EnableControls function calls the EnableCardControls function automatically. But you need to call it additionally on the following events:

OnCardWait
OnCardReady
OnReaderSelected

And additionally you need to call the EnableCardControls function each time when the active reader name of your application will be changed. In the demo application this function additionally calls on the ReadersList->OnChange event.

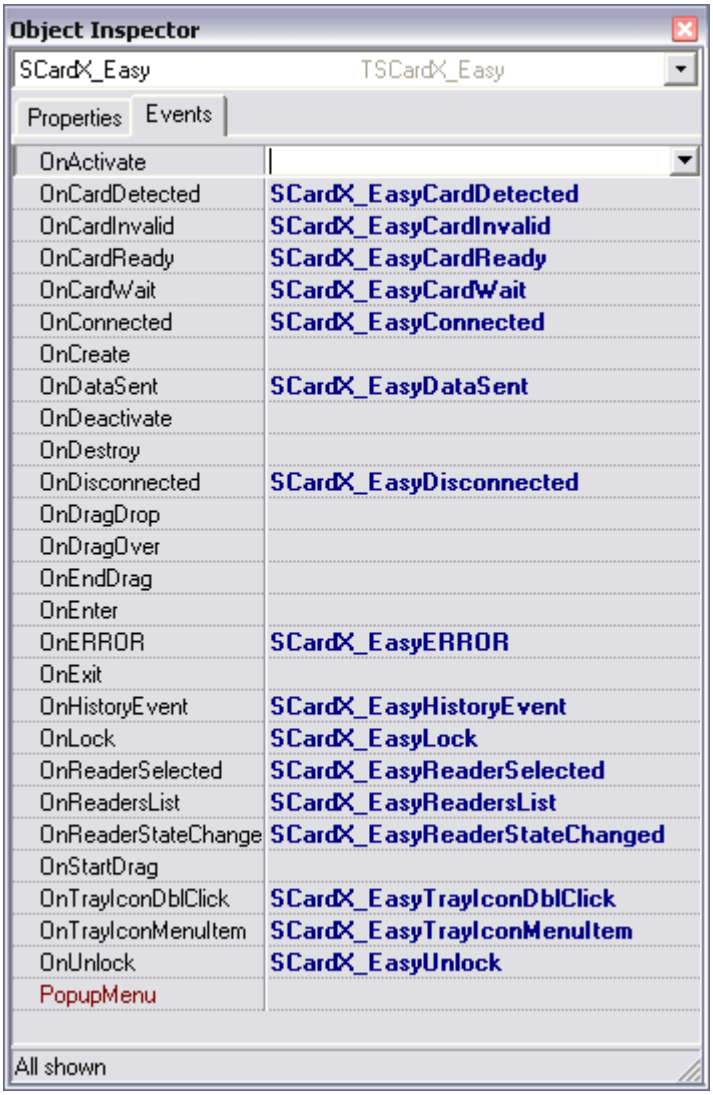
4.4 Events

How to receive the smart cards' or the service's events into your own application?

It's so easy by using of the SCardX Easy ActiveX control!

Just select the SCardX_Easy in the Object Inspector and go to the its Events page. There are all our control's events there. Double click on the event which you need and the C++ Builder will create the source code for you automatically.

Here all used in the demo application events:



All SCardX Easy ActiveX control events are maximal informative. There are all info which you need are present in the event's parameters.

For example OnCardReady event :

```
void __fastcall TMainForm::SCardX_EasyCardReady(TObject *Sender,
      BSTR *ReaderName, BSTR *ATR, long *ProtocolValue, BSTR *Protocol)
{
// .... your code here
}
```

The OnCardReady event gives you all useful information about the opened card at once:

- the opened card's Reader Name;
- the ATR of the opened card;
- the real active Protocol of the opened card.

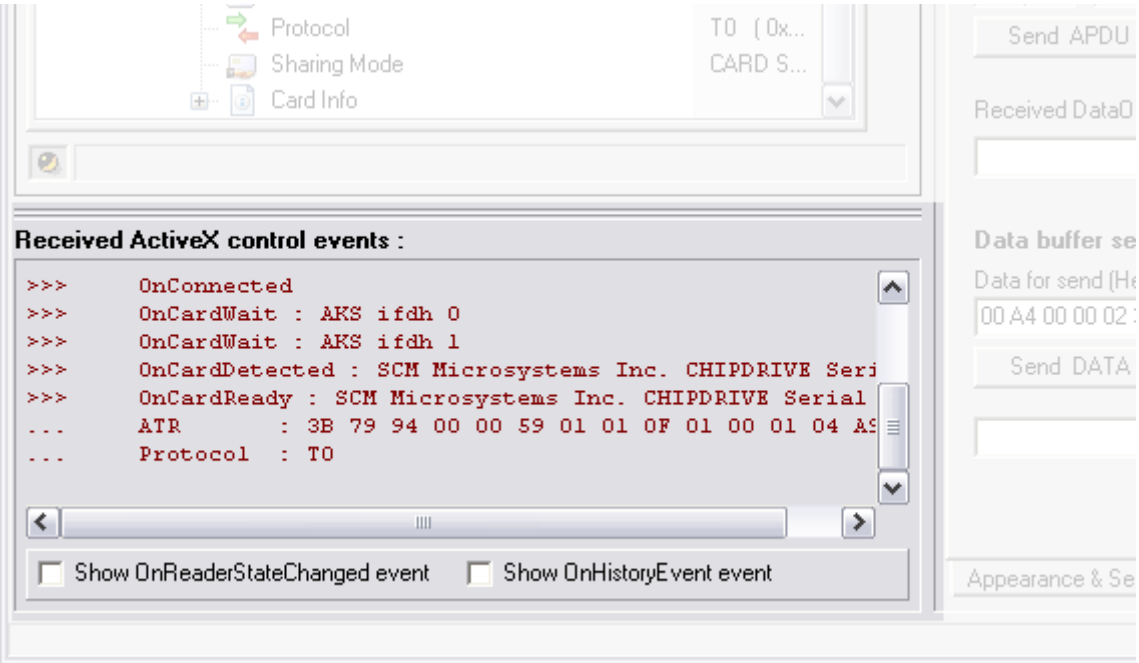
Any smart card application without the events are dead and unusable.

Otherwise by using the SCardX Easy ActiveX control you can add to your programs the power and sensitivity of the professional applications.

Processing of the received events

Each event has its own parameters list and each event is intended for its own task.

Additionally to the specific events' tasks the demo application has special controls and procedures for the simple visualization of all received events :



These events visualization tools are the Memo control and the AddEvent function:

```
void __fastcall TMainForm::AddEvent(AnsiString eType, AnsiString eMessage)
{
  AnsiString ss="";
  if (eMessage>"")
     {
     ss=" : "+eMessage;
     }
     else
     {
     ss="";
     }
  Memo->Lines->Add(">>> "+eType+ss);
}
```

It's easy! Call this function on the each occured event and you will see all received events by looking thru the text lines into the Memo control:

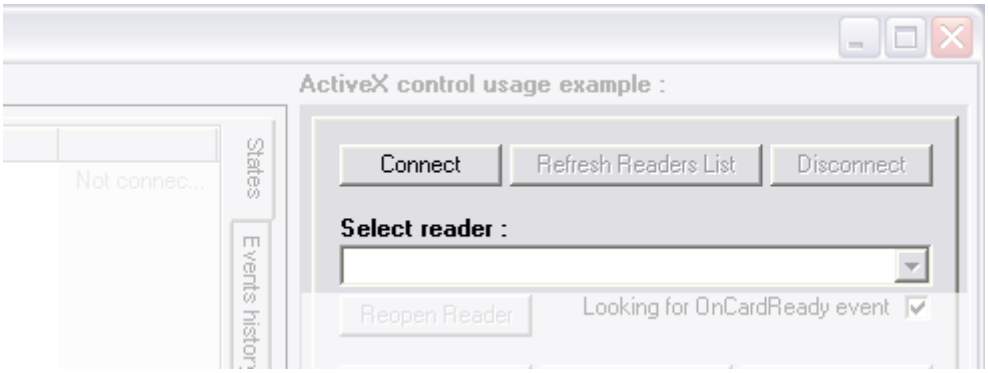
```
void __fastcall TMainForm::SCardX_EasyReaderStateChanged(TObject *Sender,
      BSTR *ReaderName, long *ReaderState, BSTR *ReaderStateHex,
```

```
BSTR *ReaderStateLookup)
{
  if (CheckBox6->Checked)
          {
          AnsiString sReaderName=*ReaderName;
          AnsiString sReaderStateHex=*ReaderStateHex;
          AddEvent ("OnReaderStateChanged",sReaderName+"  : 0x"+sReaderStateHex);
          };
}
```

4.5 Preparing the connection controls

Before working with smart cards we need to connect the smart card service.

Place on the form the three buttons for the connection commands and one combo box for the readers names list like on this picture:



Service connecting commands

By clicking on the "Connect " command button we'll select the MS Smart Card service and set up the connection state of SCardX Easy as connected:

```
void __fastcall TMainForm::Button2Click(TObject *Sender)
{
  try
  {
  SCardX_Easy->SmartCardService=srv_MS_PCSC_SCard_Service;
  SCardX_Easy->ConnectionState=stServiceConnected;
  }
  __finally
  {
  }
}
```

By clicking on the "Disconnect" command button we'll close the connection and unload the driver:

```
void __fastcall TMainForm::Button4Click(TObject *Sender)
{
  try
  {
  SCardX_Easy->ConnectionState=stServiceNotConnected;
  SCardX_Easy->SmartCardService=srv_Not_Defined;
  }
  __finally
  {
  }
}
```

What will be happened after clicking on the "Connect" button :

- the SCardX Easy loads the driver libraries and makes the connection to the selected smart card service;
- OnConnected events occurs; on this event you can enable the controls on the form by calling the EnableControls function;
- the SCardX Easy loads from the service the list of the names of the available card readers

which are attached to your PC;

- OnReadersList events occurs; on this event you can receive and store the readers list;
- the SCardX Easy starts to listen the devices for the changes of its states;
- from now the application will receive the following readers states events:
  OnReaderStateChanged
  OnCardWait : on this event you can enable the controls on the form by calling the EnableCardControls function;
  OnCardDetected
  OnCardInvalid
  OnCardReady : on this event you can enable the controls on the form by calling the EnableCardControls function;
  OnReaderSelected : on this event you can enable the controls on the form by calling the EnableCardControls function;

Readers list receiving

All smart card or device commands of the SCardX Easy ActiveX control needs the reader name as a parameter.

You can receive and store on the form the readers list by the two ways:

- using the OnReadersList event;
- using the GetReadersList function of the SCardX Easy;

The demo application uses the ReadersList combo box as a readers names' container. And additionally the selected reader of this combo box always used as the active reader name for all smart cards' and devices' commands.

For filling up of the ReadersList combo box by the real names of attached readers the demo application has a function MakeReadersList:

```
void __fastcall TMainForm::MakeReadersList(AnsiString RList)
{
  try
  {
  ReadersList->Clear();
  if (RList>"")
     {
     ReadersList->Items->Text=RList;
     ReadersList->Sorted=true;
     ReadersList->ItemIndex=0;
     };
  }
  __finally
  {
  EnableCardControls();
  }
}
```

The demo application calls the MakeReadersList automatically on the OnReadersList event:

```
void __fastcall TMainForm::SCardX_EasyReadersList(TObject *Sender,  BSTR *ReadersList)
{
  try
  {
  AnsiString sReadersList = *ReadersList;
  MakeReadersList(sReadersList);
  AddEvent("OnReadersList","");
  }
  __finally
  {
  }
}
```

It's easy! The SCardX Easy ActiveX control gives you the readers list as a parameter of the OnReadersList event!

Alternatively you can receive the readers list at any time using the GetReadersList function of the SCardX Easy. For this command the demo application has the "Refresh Readers List" button.

By clicking on the "Refresh Readers List" command button the application reloads the readers list:

```
void __fastcall TMainForm::Button3Click(TObject *Sender)
{
  try
  {
  MakeReadersList(SCardX_Easy->GetReadersList());
  }
  __finally
  {
  }
}
```

4.6 Preparing the opened reader controls

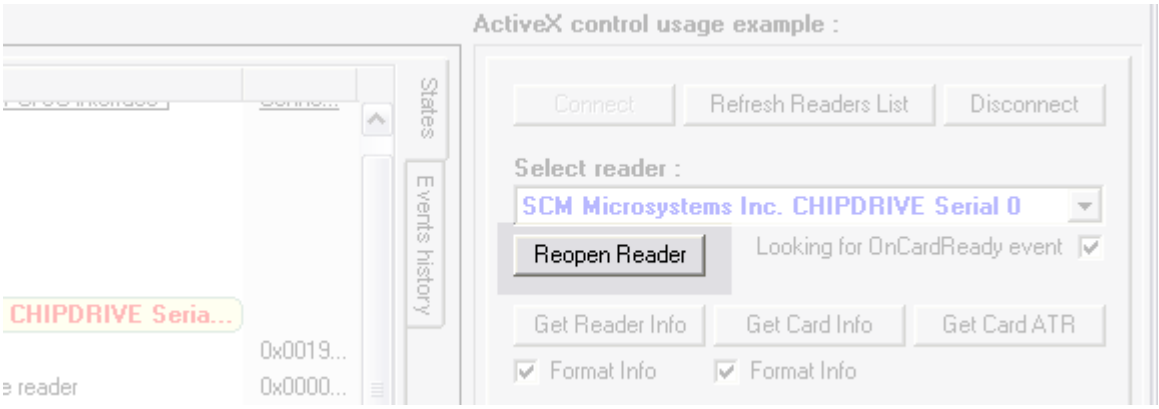
After receiving of the OnCardReady event the application may call the following functions of the SCardX Easy ActiveX control:

- ReopenReader
- GetReaderInfoFmt
- GetReaderInfo
- GetCardInfoFmt
- GetCardInfo
- GetCardATR
- SendCardAPDU
- SendCardDATA

All these functions takes the opened reader name as a parameter and may be called after receiving the OnCardReady event only.

Reopen Reader command

Add the "Reopen Reader" button on the form.

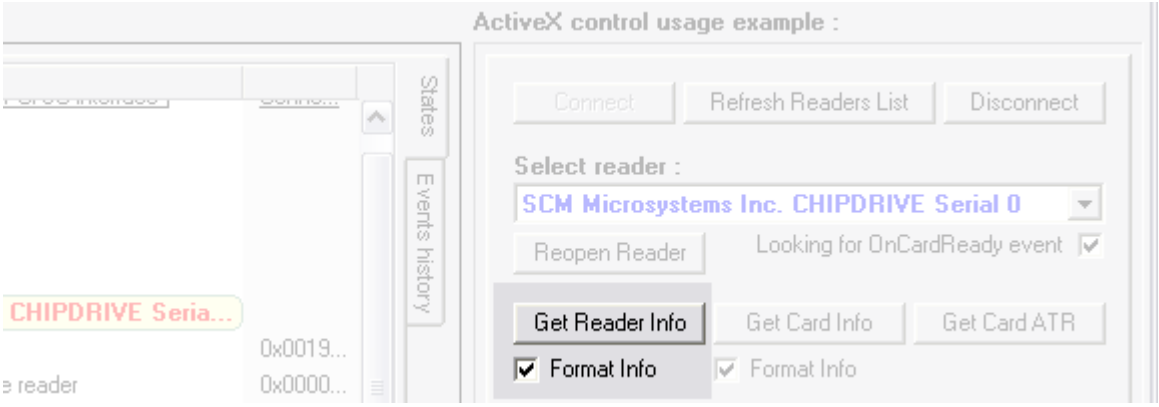


By clicking on this button the application will reopens the selected card reader:

```
void __fastcall TMainForm::Button5Click(TObject *Sender)
{
  try
  {
  WideString ww=ReadersList->Text;
  SCardX_Easy->ReopenReader(&ww);
  }
  __finally
  {
  }
}
```

Receiving the Reader Info

Add the "Get Reader Info" button and the "Format Info" checkbox on the form.



The SCardX Easy has two functions for the reader info receiving:

- GetReaderInfo
- GetReaderInfoFmt

The function GetReaderInfo returns the not formatted info lines like these ones:

```
[VENDOR INFO]
VENDOR NAME=SCM Microsystems Inc.
VENDOR IFD TYPE=CHIPDRIVE Serial
VENDOR IFD VERSION=< no info >
VENDOR IFD SERIAL NO=12639860
```

The function **GetReaderInfoFmt** returns the formatted info lines like these ones:

```
VENDOR INFO
  VENDOR NAME ..................    SCM Microsystems Inc.
  VENDOR IFD TYPE ..............    CHIPDRIVE Serial
  VENDOR IFD VERSION ...........    < no info >
  VENDOR IFD SERIAL NO .........    12639860
```

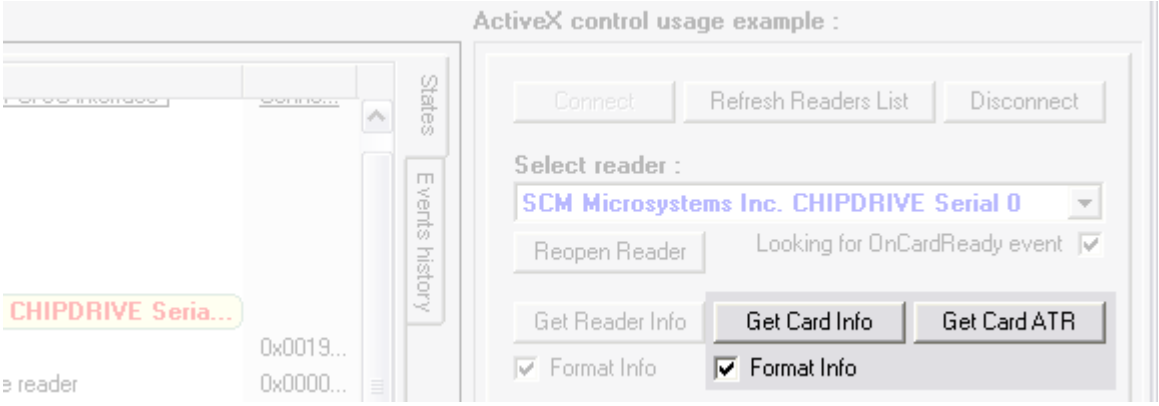
By clicking on the "Get Reader Info" button the application will receive the info lines :

```
void __fastcall TMainForm::Button6Click(TObject *Sender)
{
  try
  {
  WideString ws=ReadersList->Text;
  AnsiString s="";
  if (CheckBox8->Checked)
     { s=SCardX_Easy->GetReaderInfoFmt(&ws);}
     else
     { s=SCardX_Easy->GetReaderInfo(&ws);};
  AddEvent ("GetReaderInfo",ReadersList->Text);
  Memo->Lines->Add(s);
```

```
}
  __finally
  {
  }
}
```

Receiving the Card Info

Add the "Get Card Info" and "Get Card ATR" button and the "Format Info" checkbox on the form.



The SCardX Easy has two functions for the card info receiving:

- GetCardInfo
- GetCardInfoFmt

The function GetCardInfo returns the not formatted info lines like these ones:

```
[ICC STATE]
ATR STRING=3B 79 94 00 00 59 01 01 0F 01 00 01 04 A9
ICC PRESENCE=2
ICC INTERFACE STATUS=255
ICC TYPE PER ATR=1
CURRENT IO STATE=< no info >
[PROTOCOL]
DEFAULT DATA RATE=9624
MAX DATA RATE=115484
ASYNC PROTOCOL TYPES=3
DEFAULT CLK=3580
```

The function GetCardInfoFmt returns the formatted info lines like these ones:

```
ICC STATE
  ATR STRING ...................    3B 79 94 00 00 59 01 01 0F 01 00 01 04 A9
  ICC PRESENCE .................    2
  ICC INTERFACE STATUS .........    255
  ICC TYPE PER ATR .............    1
  CURRENT IO STATE .............    < no info >

PROTOCOL
  DEFAULT DATA RATE ............    9624
  MAX DATA RATE ................    115484
  ASYNC PROTOCOL TYPES .........    3
  DEFAULT CLK ..................    3580
```

By clicking on the "Get Card Info" button the application will receive the info lines :

```
void __fastcall TMainForm::Button7Click(TObject *Sender)
{
  try
  {
  WideString ws=ReadersList->Text;
  AnsiString s="";
  if (CheckBox9->Checked)
```

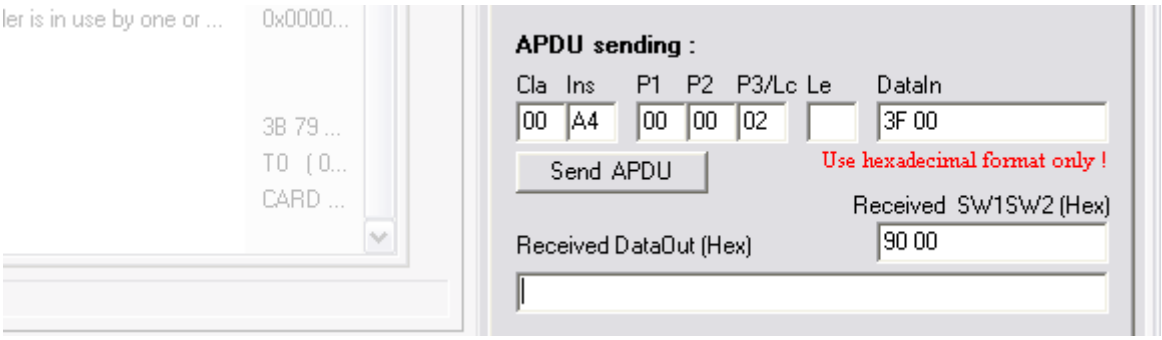
```
{ s=SCardX_Easy->GetCardInfoFmt (&ws);}
     else
     { s=SCardX_Easy->GetCardInfo (&ws);};
  AddEvent ("GetCardInfo",ReadersList->Text);
  Memo->Lines->Add(s);
  }
  __finally
  {
  }
}
```

By clicking on the "Get Card ATR" button the application will receive the ATR string of the opened card:

```
void __fastcall TMainForm::Button8Click(TObject *Sender)
{
  try
  {
  WideString ws=ReadersList->Text;
  AnsiString s=SCardX_Easy->GetCardATR (&ws);
  AddEvent ("GetCardATR",ReadersList->Text);
  Memo->Lines->Add(s);
  }
  __finally
  {
  }
}
```

Command APDU sending

Add on the form the following controls:



By clicking on the "Send APDU" button the application gets the hexadecimal parts of a command APDU according to ISO-7816 from the form's edit controls and puts its to the parameters of the SCardX Easy's function SendCardAPDU :

```
void __fastcall TMainForm::Button9Click(TObject *Sender)
{
SW->Text="";
DataOut->Text="";
AnsiString ss="";

WideString wReaderName     = ReadersList->Text;
WideString wCla            = Cla->Text;
WideString wIns            = Ins->Text;
WideString wP1             = P1->Text;
WideString wP2             = P2->Text;
WideString wP3             = P3->Text;
WideString wLe             = Le->Text;
WideString wDataIn         = DataIn->Text;
WideString wSW             = "";
WideString wDataOut        = "";

  try
  {
```

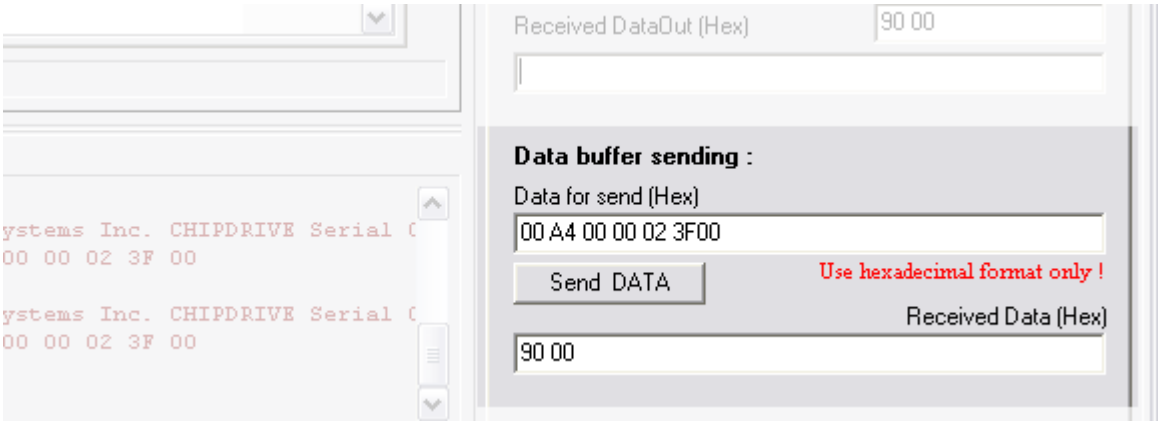
```
try
    {
    ss=SCardX_Easy-
>SendCardAPDU(&wReaderName,&wCla,&wIns,&wP1,&wP2,&wP3,&wLe,&wDataIn,&wSW,&wDataOut);
    }
    catch (Exception &exception)
    {
    ss="";
    ShowMessage(exception.Message);
    }
  }
  __finally
  {
  if ( ss>"")
     {
     SW->Text=wSW;
     DataOut->Text=wDataOut;
     };
  }
}
```

This function returns the card's response APDU data as parameters according to ISO-7816 :

- the status word SW1SW2 value in the hexadecimal format; it placed in the edit control labeled "Received SW1SW2 (Hex)";
- the DataOut buffer in the hexadecimal format; it placed in the edit control labeled "Received DataOut (Hex)"

Unformatted data buffers sending

Add on the form the following controls:



By clicking on the "Send DATA" button the application gets the hexadecimal value of the send buffer labeled as "Data for send (Hex)" and puts its to a parameter of the SCardX Easy's function SendCardDATA :

```
void __fastcall TMainForm::Button10Click(TObject *Sender)
{
DataReceived->Text="";
AnsiString ss="";

WideString wReaderName     = ReadersList->Text;
WideString wDataSend       = DataSend->Text;
AnsiString sDataReceived   = "";

  try
  {
    try
    {
    sDataReceived  = SCardX_Easy->SendCardDATA(&wReaderName,&wDataSend);
    }
    catch (Exception &exception)
```

```
{
    sDataReceived="";
    ShowMessage(exception.Message);
    }
  }
  __finally
  {
  if ( sDataReceived>"")
     {
     DataReceived->Text=sDataReceived;
     };
  }
}
```

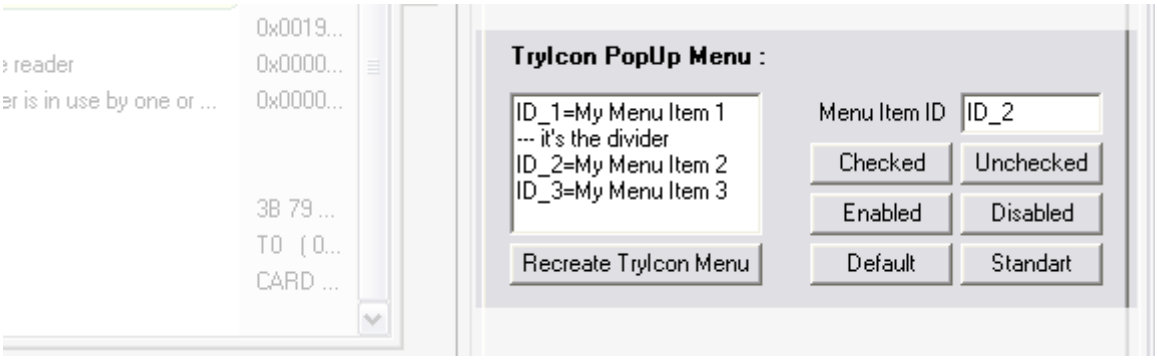
This function returns the card's answer on the sent data in the hexadecimal format. The returned data placed in the edit control labeled "Received Data (Hex)".

4.7 Tray Icon

The SCardX Easy ActiveX control allows you to manage the tray icon pop-up menu items and to receive the tray icon events.

Preparing the form controls

Add on the form the following controls:



Creating your own tray icon menu

The new pop-up menu of the SCardX Easy tray icon creates easy by calling of the TrayIconMenuCreate function.

You need to prepare the menu items list according to these rules:

- each new line in the list is the new menu item template;
- each menu item template consists of two parts;
  - the menu item ID;
  - the menu item caption;
- the parts of the menu item template are divided by the "=" character;

- if the menu item template begins with a "-" character the menus divider will be created;

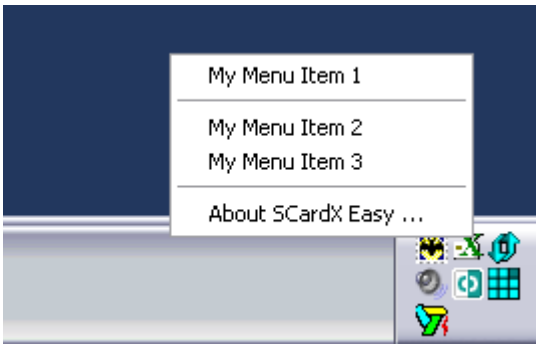
Use the memo on the form for preparing of the menu items list before menu creating.

For example your menu items list may be prepared like this:

```
• ID_1=My Menu Item 1
• - it's the divider
• ID_2=My Menu Item 2
• ID_3=My Menu Item 3
```

By clicking on the "Recreate TryIcon Menu" button the SCardX Easy will recreate its tray icon pop-up menu:

```
void __fastcall TMainForm::Button14Click(TObject *Sender)
{
  try
  {
  WideString ww=MenuItemsList->Lines->Text;
  SCardX_Easy->TrayIconMenuCreate(&ww);
  }
  __finally
  {
  }
}
```



Changing the menu item properties

You can set up the following menu item properties:

- checked / unchecked
- enabled / disabled
- default / standart

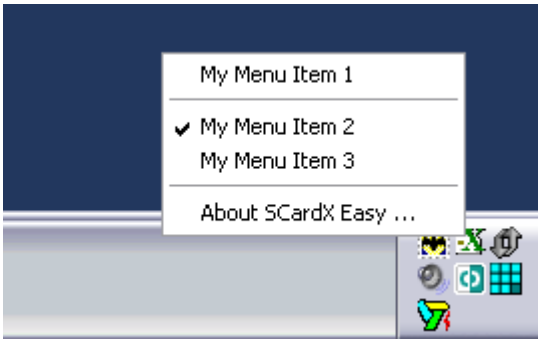
All functions for changing of the menu item's properties takes the item ID string as a parameter.

Setting up the menu item as checked / unchecked

By clicking on the "Checked" button the SCardX Easy makes the menu item as checked:

```
void __fastcall TMainForm::Button18Click(TObject *Sender)
{
  try
  {
  WideString ww=Edit3->Text;
  VARIANT_BOOL bb=true;
  SCardX_Easy->TrayIconMenuItemSetChecked(&ww,&bb);
```

```
}
  __finally
  {
  }
}
```



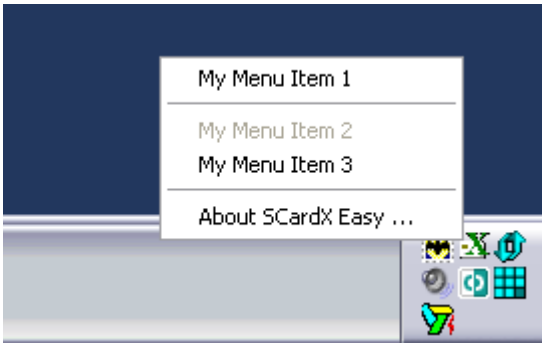
By clicking on the "Unchecked" button the SCardX Easy makes the menu item as unchecked:

```
void __fastcall TMainForm::Button16Click(TObject *Sender)
{
  try
  {
  WideString ww=Edit3->Text;
  VARIANT_BOOL bb=false;
  SCardX_Easy->TrayIconMenuItemSetChecked(&ww,&bb);
  }
  __finally
  {
  }
}
```

Setting up the menu item as enabled / disabled

By clicking on the "Disabled" button the SCardX Easy makes the menu item as disabled:

```
void __fastcall TMainForm::Button15Click(TObject *Sender)
{
  try
  {
  WideString ww=Edit3->Text;
  VARIANT_BOOL bb=false;
  SCardX_Easy->TrayIconMenuItemSetEnabled(&ww,&bb);
  }
  __finally
  {
  }
}
```



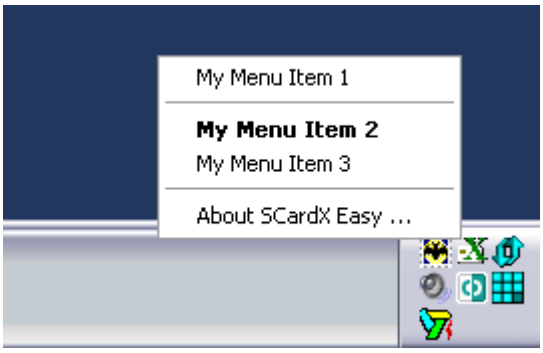
By clicking on the "Enabled" button the SCardX Easy makes the menu item as enabled:

```
void __fastcall TMainForm::Button19Click(TObject *Sender)
{
  try
  {
  WideString ww=Edit3->Text;
  VARIANT_BOOL bb=true;
  SCardX_Easy->TrayIconMenuItemSetEnabled(&ww,&bb);
  }
  __finally
  {
  }
}
```

Setting up the menu item as default / standart

By clicking on the "Default" button the SCardX Easy makes the menu item as default:

```
void __fastcall TMainForm::Button20Click(TObject *Sender)
{
  try
  {
  WideString ww=Edit3->Text;
  VARIANT_BOOL bb=true;
  SCardX_Easy->TrayIconMenuItemSetDefault(&ww,&bb);
  }
  __finally
  {
  }
}
```



By clicking on the "Standart" button the SCardX Easy makes the menu item as standart menu item:

```
void __fastcall TMainForm::Button17Click(TObject *Sender)
{
  try
  {
  WideString ww=Edit3->Text;
  VARIANT_BOOL bb=false;
  SCardX_Easy->TrayIconMenuItemSetDefault(&ww,&bb);
  }
  __finally
  {
  }
}
```

Receiving the tray icon menu events

The SCardX Easy creates the OnTrayIconMenuItem event when user clicks on the menu item:

```
void __fastcall TMainForm::SCardX_EasyTrayIconMenuItem(TObject *Sender,
      BSTR *ItemID, VARIANT_BOOL *IsChecked, VARIANT_BOOL *IsEnabled,
      VARIANT_BOOL *IsDefault, BSTR *Caption)
{
  AddEvent ("OnTrayIconMenuItem",*Caption);
}
```

Receiving the tray icon mouse double click event

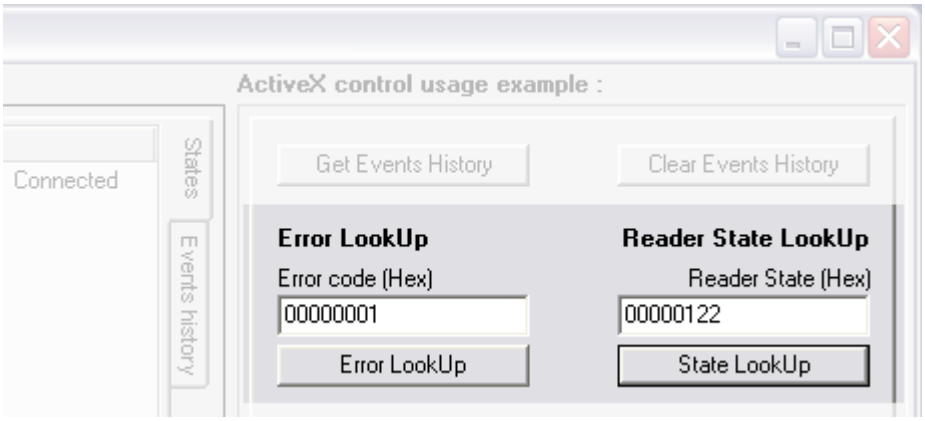
The SCardX Easy creates the OnTrayIconDblClick event when user double clicks on the tray icon:

```
void __fastcall TMainForm::SCardX_EasyTrayIconDblClick(TObject *Sender)
{
  AddEvent ("OnTrayIconDblClick","");
}
```

4.8 LookUp service

The SCardX Easy allows you to decode the system error codes and the reader states codes from its numerical values to its string descriptions.

Add on the form the following controls:



Error LookUp

By clicking on the "Error LookUp" button the application calls the lookup function and receives the decoded value:

```
void __fastcall TMainForm::Button12Click(TObject *Sender)
{
  try
  {
  WideString ww=Edit1->Text;

  AnsiString ss=SCardX_Easy->LookUpError(&ww);

  AddEvent ("LookUpError","");
  Memo->Lines->Add ("0x"+Edit1->Text+" : "+ss);
  }
  __finally
  {
  }
}
```

Reader State LookUp

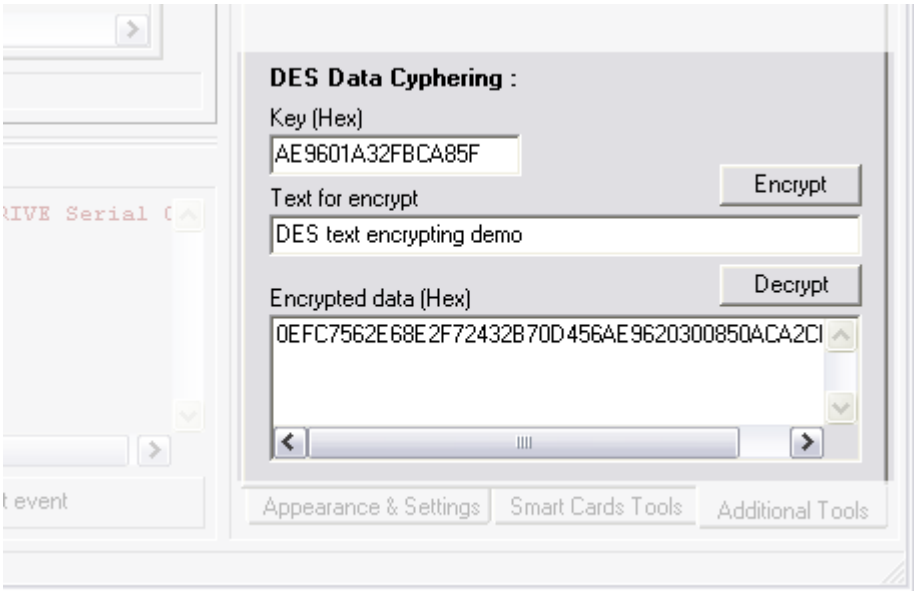
By clicking on the "State LookUp" button the application calls the lookup function and receives the decoded value:

```
void __fastcall TMainForm::Button13Click(TObject *Sender)
{
  try
  {
  WideString ww=Edit2->Text;

  AnsiString ss=SCardX_Easy->LookUpReaderState(&ww);

  AddEvent ("LookUpReaderState","0x"+Edit2->Text);
  Memo->Lines->Add (ss);
  }
  __finally
  {
  }
}
```

4.9 Data ciphering

The SCardX Easy ActiveX control allows you to encrypt and to decrypt the text strings using the DES algorithm.

Add on the form the following controls:



Before using the ciphering functions you need to prepare the Key value in the hexadecimal format.

Key rules:

- if you want to use ASCII symbols like the letters or numbers as a key you need to convert its char codes to a hexadecimal format;

  for example the ASCII text "**MyKey123** " in the hex format is "**4D794B6579313233** "

- the length of the binary key always must be 8 bytes and the length of the key in the hexadecimal format always must be 16 hex symbols!

Create the new key and place its hex value into the edit control labeled "Key (Hex)".

DES data encoding

Type any text you like into the text control labeled "Text for encrypt".

By clicking on the "Encrypt" button the application takes the key hex value and the text for encrypt from the form's edit controls and calls the DES_EncryptString encrypt function:

```
void __fastcall TMainForm::Button21Click(TObject *Sender)
{
  try
  {
  WideString kk=Trim(KeyDES->Text);
  WideString ww=Trim(EncText->Text);

  AnsiString ss=SCardX_Easy->DES_EncryptString(&kk,&ww);

  AddEvent ("DES Encrypt String","");
  Memo->Lines->Add ("DES Key          : "+kk);
  Memo->Lines->Add ("Text for encrypt : "+ww);
```

```
Memo->Lines->Add ("Encrypted data    : "+ss);
  }
  __finally
  {
  }

}
```

Encrypting example:

```
DES Key :            AE9601A32FBCA85F
Text :               Demo text for encrypt
Encrypted data :     D6 D1 DB 24 59 8B 3A 9F 4D 22 58 96 68 92 AB 29 40 41 16 B4 69 64 15
28
```

DES data decoding

Type the previous encrypted text as a hex buffer into the text control labeled "Encrypted data (Hex)".

By clicking on the "Decrypt" button the application takes the key hex value and the encrypted hex buffer from the form's edit controls and calls the DES_DecryptString decrypt function:

```
void __fastcall TMainForm::Button22Click(TObject *Sender)
{
  try
  {
  WideString kk=Trim(KeyDES->Text);
  WideString ww=Trim(DecText->Lines->Text);

  AnsiString ss=SCardX_Easy->DES_DecryptString(&kk,&ww);

  AddEvent ("DES Decrypt String","");
  Memo->Lines->Add ("DES Key          : "+kk);
  Memo->Lines->Add ("Encrypted data   : "+ww);
  Memo->Lines->Add ("Decrypted data   : "+ss);
  }
  __finally
  {
  }
}
```

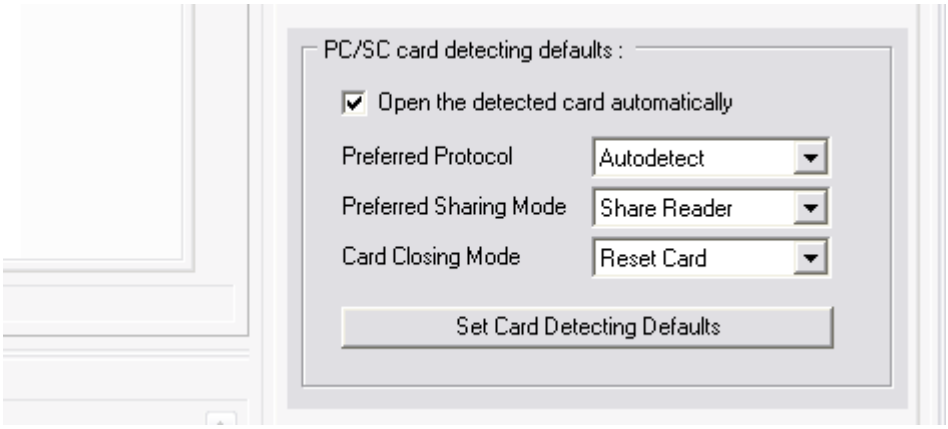
Decrypting example:

```
DES Key :            8CA64DE9C1B123A7
Encrypted data :     BA 40 AC 43 81 34 9A DC AF 60 0B D5 EC 49 86 F8 90 7B B0 71 C1 05 38
A9
Decrypted text :     Decrypt demo text
```

4.10 Card detecting defaults

The SCardX Easy allows you to set up the card detecting defaults.

Add on the form the following controls:



Fill up the combo box labeled "Preferred Protocol" by the following string list:

- Autodetect
- T0
- T1
- RAW
- Undefined

Fill up the combo box labeled "Preferred Sharing Mode" by the following string list:

- Share Reader
- Exclusive Use
- Direct Reader Control

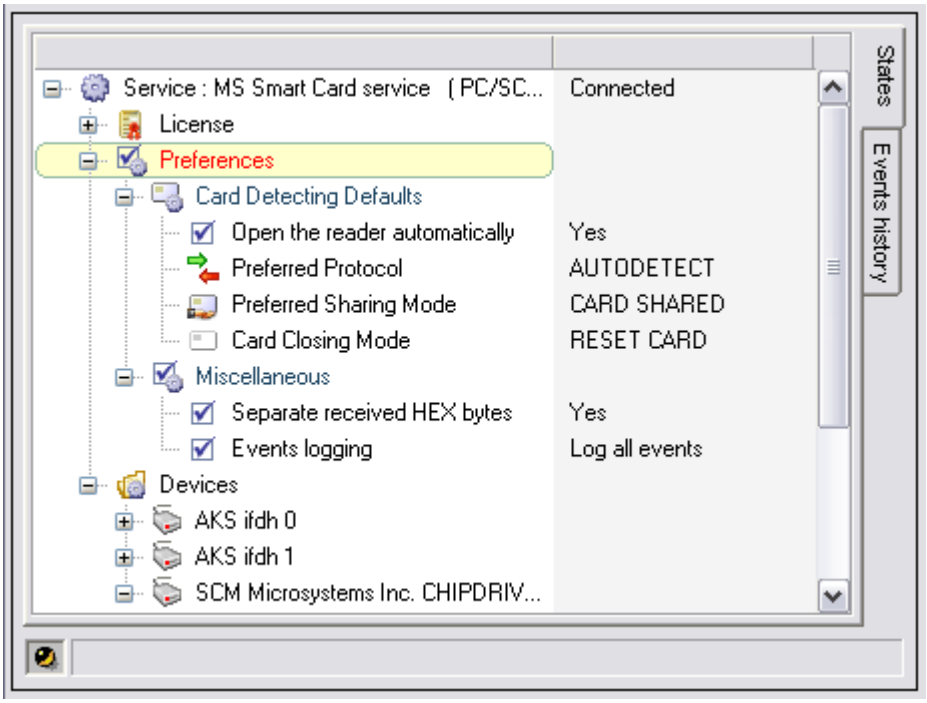
Fill up the combo box labeled "Card Closing Mode" by the following string list:

- Leave Card
- Reset Card
- Unpower Card
- Eject Card

By clicking on the "Set Card Detecting Defaults" button the application sets up the preferences values using the SetPref_PCSC_OnCardDetect function:

```
void __fastcall TMainForm::Button23Click(TObject *Sender)
{
  try
  {

  VARIANT_BOOL AutoOpen=CheckBox12->Checked;
  TxProtocol PreferredProtocol= TxProtocol(ComboBox1->ItemIndex);
  TxSharingMode PreferredSharingMode= TxSharingMode(ComboBox2->ItemIndex);
  TxCardClosingMode CardClosingMode= TxCardClosingMode(ComboBox3->ItemIndex);

  SCardX_Easy->SetPref_PCSC_OnCardDetect( &AutoOpen,
                                          &PreferredProtocol,
                                          &PreferredSharingMode,
                                          &CardClosingMode);
  }
  __finally
  {
  }

}
```

All preferences changes becomes visible on the "States" page of the SCardX Easy immediately:



4.11 Configuring the application startup

The application startup is a good moment for setting up the SCardX Easy's properties.

```
// setting up the user interface properties
SCardX_Easy->VisibleToolBar=false;
SCardX_Easy->VisibleStatusBar=true;
SCardX_Easy->EventsHistoryEnabled=true;
SCardX_Easy->VisibleEventsHistory=true;
SCardX_Easy->ActivePage=apStates;

// connecting the service
SCardX_Easy->SmartCardService=srv_MS_PCSC_SCard_Service;
SCardX_Easy->ConnectionState=stServiceConnected;
```

We recommend you to set up the user interface properties of the SCardX Easy like the BorderStyle and other on the application startup.

Additonally you may call the interface function:

```
// enabling/disabling the controls
EnableControls();
```

4.12 Configuring the application shutdown

Important! Be careful!

You must call the finalization function of the SCardX Easy on the application's shutdown!

```
void __fastcall TMainForm::FormClose(TObject *Sender, TCloseAction &Action)
{
  try
  {
  SCardX_Easy->Finalize();
  }
  __finally
  {
  }
}
```

4.13 Tell : - " Hello, cards World ! "

Ok. Your first application is already prepared and ready to start!

ISO-7816 standard and smart card basics

The ISO-7816 is a base of the smart cards functionality. All another smart cards specifications was created under this standard and expands it only.

The card command may be sent into a card as a data buffer which is formatted as a **command APDU** (**A**pplication **P**rotocol **D**ata **U**nit).

The card's answer on each **command APDU** is the data buffer which is formatted as a **response APDU** .

According to ISO-7816-4 5.3.1 the **command APDU** consists of :

- a mandatory header of 4 bytes : **Cla Ins P1 P2** ;
- a conditional body of a variable length;

Command APDU structure:

Header	Body
Cla Ins P1 P2	[**Lc** field] [**DataIn** field] [**Le** field]

What is the **command APDU** content?

According to ISO-7816-4 5.4 the **command APDU** contents :

Code	Name	Length	Description
Cla **Ins** **P1** **P2**	Class Instruction Parameter1 Parameter 2	**1** **1** **1** **1**	Class of instruction Instruction code Instruction parameter 1 Instruction parameter 2
P3/Lc field	Length	variable **1** or **3**	Number of bytes present in the data field of the command
DataIn field	Data	variable **= Lc**	String of bytes sent in the data field of the command
Le field	Length	variable **<= 3**	Maximum number of bytes inspected in the data field of the response to the command

So each command is an APDU-formatted array of bytes which may be sent into a card.

What happens after the data was sent?

The card answers on the sent command APDU by its **response APDU** .

According to ISO-7816-4 5.3.3 the **response APDU** consists of :

- a conditional body of a variable length;
- a mandatory trailer of 4 bytes (status word) : **SW1 SW2** ;

Body	Trailer
[**DataOut field**]	**SW1 SW2**

What is the **response APDU** content?

According to ISO-7816-4 5.4 the **response APDU** contents :

Code	Name	Length	Description
DataOut field	Data	variable **= Le**	String of bytes received in the data field of the response
SW1 **SW2**	Status byte 1 Status byte 2	**1** **1**	Command processing status Command processing qualifier

How it works?

For preparing of the command you need only to fill up the **command APDU** fields according to the card command which you need send into the card. Where can you find the values of these fields? You may find all necessary info about the **command APDU** and **response APDU** fields' values in the specifications of your smart cards.

The ISO-7816 standard defines the global principles of the card's functionality only.

The real cards always differs by its available commands' set and by the values of the **command APDU** fields and all cards differs by its **response APDU** fields values.

But all chip smart cards always receives the commands as **command APDU** 's and answers back by the **response APDU** 's according to ISO-7816.

Please look more about the smart cards basics into the ISO-7816 standard and into the your cards' specifications.

Your first smart card command

As example we'll use the GSM SIM card and the GSM11.11 card specification.

According to ISO-7816 any chip smart card must have the Master File (MF) named 3F00. It's the "root directory" of the smart card's filesystem. The SIM card has the "3F00" file too.

We'll try to send to the SIM card the command SELECT MF.

According to GSM11.11 9.2.1 the **command APDU** for the command SELECT is defined as:

9.2.1 SELECT

COMMAND	CLASS	INS	P1	P2	P3
SELECT	'A0'	'A4'	'00'	'00'	'02'

Command parameters/data:

Byte(s)	Description	Length
1 - 2	File ID	2

And according to GSM11.11 9.4.1 the successful **respond APDU** is defined as:

9.4 Status conditions returned by the card

This subclause specifies the coding of the status words SW1 and SW2.

9.4.1 Responses to commands which are correctly executed

SW1	SW2	Description
'90'	'00'	- normal ending of the command
'91'	'XX'	- normal ending of the command, with extra information from the proactive SIM containing a command for the ME. Length 'XX' of the response data
'9E'	'XX'	- length 'XX' of the response data given in case of a SIM data download error
'9F'	'XX'	- length 'XX' of the response data

So, according to GSM11.11 our **command APDU** is:

```
Cla       = A0
Ins       = A4
P1        = 00
P2        = 00
P3/Lc     = 02
DataIn    = 3F00
```

And after of this **command APDU** will be sent into the card you may receive from the card the following **response APDU** :

```
DataOut      = <none>
SW1 SW2      = 9F XX ( where XX is the length of the response data)
```

You can test this command using your new smart card application:

1. run the application;
2. connect to the service;
3. insert the card into a reader;
4. after the card will be opened by SCardX Easy please select your reader in the readers list on the form;
5. fill up the fields of the "**APDU Sending** " controls on the form according to the **command APDU** which was defined before;
6. click on the "**Send APDU** " button;
7. after the command sending please look on the edit control labeled as "**Received SW1 SW2 (Hex)** " - there is the status word hex value like "**9F17"** must present there;

That's all.

You have prepared your first command APDU, you have sent this command into the card and you have received from the card its answer on your command.

Congratulations!
At this moment you already have told to your SIM card - " Hello, cards World ! ".

5 SCardX Easy interface specification

5.1 Properties

User interface properties

ActivePage
BorderStyle
BorderWidth
EventsHistoryEnabled
EventsLogging
Visible
VisibleEventsHistory
VisibleStatusBar
VisibleToolBar
VisibleTrayIcon

Smart card work properties

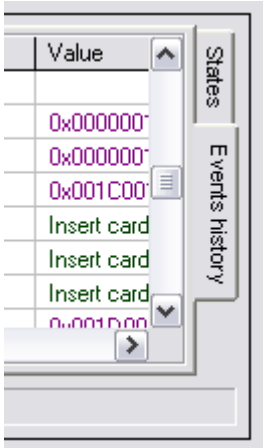
ConnectionState
SmartCardService
SeparateReceivedBytes

5.1.1 ActivePage

Specifies what the page of SCardX Easy is on the front of the control.

apStates

apEventsHistory



Description

Use the ActivePage property to determine what page is on the front of the control.

Type:

```
C++    : int
Basic  : As Long
Delphi : Integer
```

Possible values:

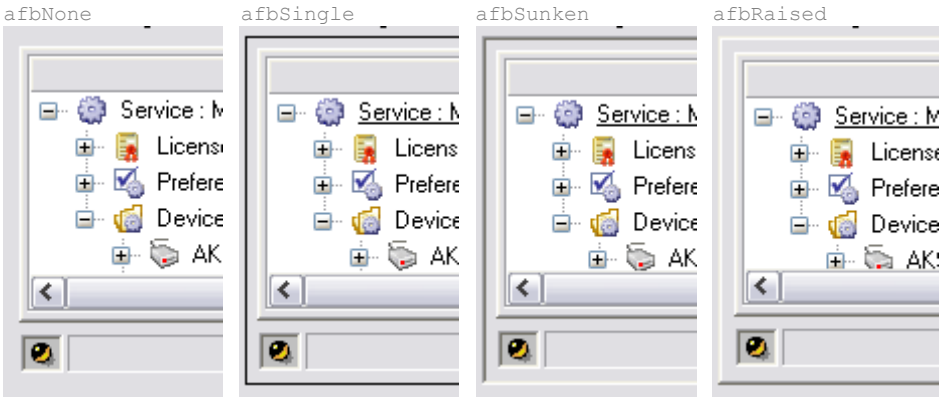
```
apStates          = $00000000
apEventsHistory   = $00000001
```

C++ Builder syntax:

```
SCardX_Easy->ActivePage=apEventsHistory;
```

5.1.2 BorderStyle

Specifies the drawing style of the border of the SCardX Easy control.



Description

Use the BorderStyle property for setting up the control's border style.

Type:

```
C++    : int
Basic  : As Long
Delphi : Integer
```

Possible values:

```
afbNone    = $00000000
afbSingle  = $00000001
afbSunken  = $00000002
afbRaised  = $00000003
```

C++ Builder syntax:

```
SCardX_Easy->BorderStyle=3;
```

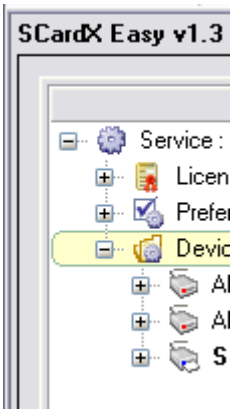
5.1.3 BorderWidth

Specifies the control's inner border width.

BorderWidth = 0

BorderWidth = 5

BorderWidth = 10



Description

Use the BorderWidth property for setting up the control's inner border width.

Type:

```
C++    : int
Basic  : As Long
Delphi : Integer
```

C++ Builder syntax:

```
SCardX_Easy->BorderWidth=15;
```

5.1.4 ConnectionState

Specifies the current state of the connection to the selected smart card service.

Description

Use the ConnectionState property for connecting or disconnecting of the selected smart card service.

This property is unavailable while the SmartCardService is equal srv_Not_Defined.

Type:

```
C++    : int
Basic  : As Long
Delphi : Integer
```

Possible values:

```
stServiceNotConnected  = $00000000
stServiceConnected     = $00000001
```

C++ Builder syntax:

```
SCardX_Easy->ConnectionState=stServiceConnected;
```

5.1.5 EventsHistoryEnabled

Specifies whether the events history logging is enabled.

Description

Use the EventsHistoryEnabled property for enabling or disabling the logging of events on the Events History page.

Type:

```
C++    : bool
Basic  : As Boolean
Delphi : WordBool
```

C++ Builder syntax:

```
SCardX_Easy->EventsHistoryEnabled=true;
```

5.1.6 EventsLogging

Specifies the events logging mode.

Description

Use the EventsLogging property to determine the control's events logging mode.

Set the EventsLogging to xLog_AllEvents if you need more detailed events log.

Type:

```
C++    : int
Basic  : As Long
Delphi : Integer
```

Possible values:

```
xLog_AllEvents         = $00000000
xLog_MostUsefulEvents  = $00000001
```

C++ Builder syntax:

```
SCardX_Easy->EventsLogging=xLog_AllEvents;
```

5.1.7 SeparateReceivedBytes

Specifies whether the received from the card hex bytes will be separated by the space character.

Description

Set the SeparateReceivedBytes property to true if you want to receive the separated bytes like this:

```
3B 79 94 00 59 01 01 0F 01
```

Otherwise the data will be received and showed like this:

```
3B799400005901010F01
```

Type:

```
C++    : bool
Basic  : As Boolean
Delphi : WordBool
```

C++ Builder syntax:

```
SCardX_Easy->SeparateReceivedBytes=true;
```

5.1.8 SmartCardService

Specifies the selected smart card service.

Description

Use this property to change the selected smart card service.

If the srv_Not_Defined value assigned in this case the SCardX Easy closes all active connections and unloads the previous loaded service's drivers.

If the srv_MS_PCSC_SCard_Service value assigned in this case the SCardX Easy tries to find the MS Smart Card service's libraries and loads its.

After the service will be loaded you can connect of this service by assigning the value stServiceConnected to the ConnectionState property.

Type:

```
C++    : int
Basic  : As Long
Delphi : Integer
```

Possible values:

```
srv_Not_Defined            = $00000000
srv_MS_PCSC_SCard_Service  = $00000001
```

C++ Builder syntax:

```
SCardX_Easy->SmartCardService=srv_MS_PCSC_SCard_Service;
```

5.1.9 Visible

Specifies the SCardX Easy control's visibility.

Description

Set the Visible property to false if you wish to hide the control on your application.

Type:

```
C++    : bool
Basic  : As Boolean
Delphi : WordBool
```

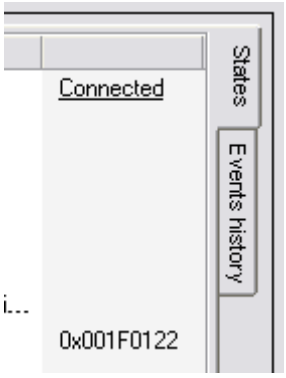
C++ Builder syntax:

```
SCardX_Easy->Visible=true;
```

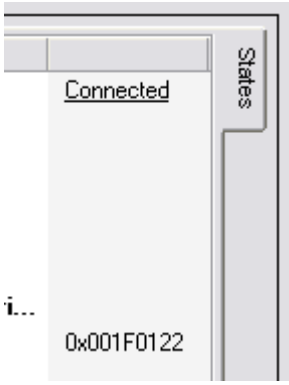
5.1.10 VisibleEventsHistory

Specifies the visibility of the "Events History" panel.

True



False



Description

Use the VisibleEventsHistory property for showing or hiding the "Events History" panel of the control.

Type:

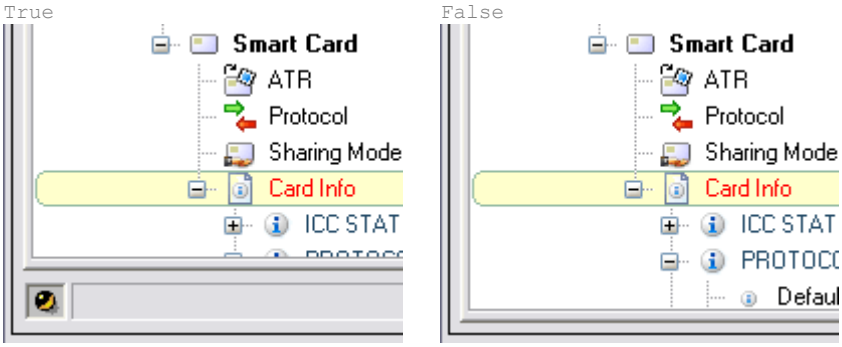
```
C++    : bool
Basic  : As Boolean
Delphi : WordBool
```

C++ Builder syntax:

```
SCardX_Easy->VisibleEventsHistory=true;
```

5.1.11 VisibleStatusBar

Specifies the visibility of the status bar of the SCardX Easy.



Description

Use the VisibleStatusBar property for showing or hiding the status bar of the control.

Type:

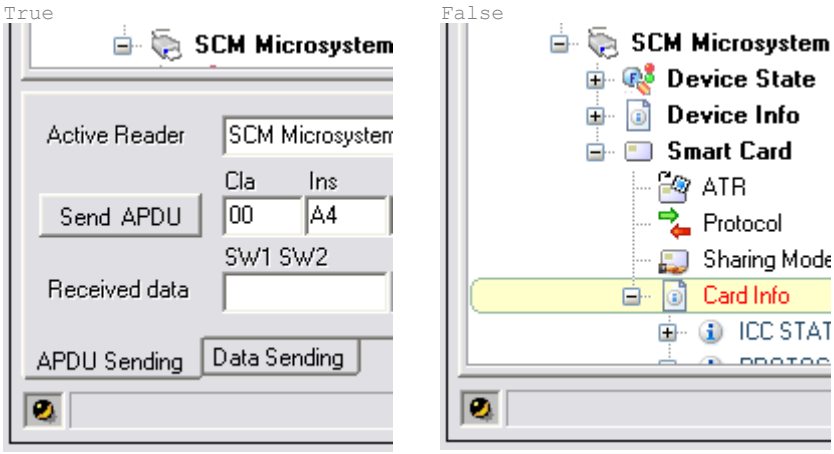
```
C++    : bool
Basic  : As Boolean
Delphi : WordBool
```

C++ Builder syntax:

```
SCardX_Easy->VisibleStatusBar=false;
```

5.1.12 VisibleToolBar

Specifies the visibility of the tool bar of the SCardX Easy.



Description

Use the VisibleToolBar property for showing or hiding the tool bar of the control.

Type:

```
C++    : bool
Basic  : As Boolean
Delphi : WordBool
```

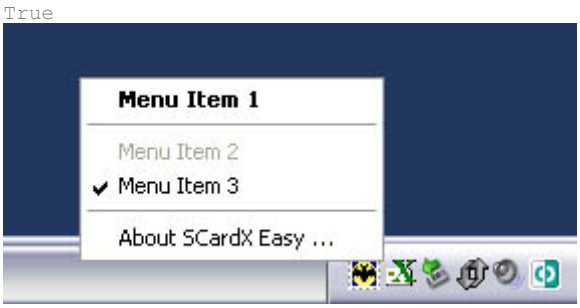
C++ Builder syntax:

```
SCardX_Easy->VisibleToolBar=false;
```

5.1.13 VisibleTrayIcon

Specifies the visibility of the tray icon of the SCardX Easy.

True



Description

Use the VisibleTrayIcon property for showing or hiding the tray icon of the control.

Warning! You can hide the TrayIcon under the Site or Developer's License only !

Type:

```
C++    : bool
Basic  : As Boolean
Delphi : WordBool
```

C++ Builder syntax:

```
SCardX_Easy->VisibleTrayIcon=false;
```

5.2 Functions

User interface functions

EventsHistoryClear

GetEventsHistory
SetPref_PCSC_OnCardDetect
TrayIconMenuClear
TrayIconMenuCreate
TrayIconMenuItemSetChecked
TrayIconMenuItemSetDefault
TrayIconMenuItemSetEnabled

Smart card work functions

GetCardATR
GetCardInfo
GetCardInfoFmt
GetReaderInfo
GetReaderInfoFmt
GetReadersList
IsCardReady
ReopenReader
SendCardAPDU
SendCardDATA

Other functions

DES_DecryptString
DES_EncryptString
Finalize
IsLocked
LookUpError
LookUpReaderState
Version
VersionMajor
VersionMinor

5.2.1 DES_DecryptString

Decrypts the encrypted by DES algorithm hexadecimal data buffer.

Arguments / parameters

Argument Name	Data Type	Description
KeyHEX (input)	C++ : BSTR Basic : As String Delphi : WideString	the hex data buffer of the DES key value; • the length of the binary key always must 8 bytes and the length of the key in the hexadecimal format always must 16 hex symbols; • do not use ASCII symbols for the key value : always use the hexadecimal format only;
EncryptedDataHEX (input)	C++ : BSTR Basic : As String Delphi : WideString	the hex data buffer of the previously encrypted by DES text string;

All arguments are passed by reference.

Returns

The function returns the decrypted text string.

Returning value data type

C++ : BSTR
Basic : As String
Delphi : WideString

C++ Builder syntax:

```
WideString kk="AE9601A32FBCA85F";
WideString ww="0EFC7562E68E2F72432B70D456AE9620300850ACA2CFC7EE";

AnsiString ss=SCardX_Easy->DES_DecryptString(&kk,&ww);
```

5.2.2 DES_EncryptString

Encrypts ASCII symbols text string by the DES algorithm.

Arguments / parameters

Argument Name	Data Type	Description
KeyHEX (input)	C++ : BSTR Basic : As String Delphi : WideString	the hex data buffer of the DES key value; • the length of the binary key always must 8 bytes and the length of the key in the hexadecimal format always must 16 hex symbols; • do not use ASCII symbols for the key value : always use the hexadecimal format only;
CryptString (input)	C++ : BSTR Basic : As String Delphi : WideString	any text string for encrypt;

All arguments are passed by reference.

Returns

The function returns the hex data buffer of the encrypted string.

Returning value data type

C++ : BSTR
Basic : As String
Delphi : WideString

C++ Builder syntax:

```
WideString kk="AE9601A32FBCA85F";
WideString ww="DES text encrypting demo";

AnsiString ss=SCardX_Easy->DES_EncryptString(&kk,&ww);
```

5.2.3 EventsHistoryClear

Deletes all events messages from the grid of the "Events History" page of the control.

Arguments / parameters

<none>

Returns

<none>

C++ Builder syntax:

```
SCardX_Easy->EventsHistoryClear();
```

5.2.4 Finalize

Closes all opened connections and frees all used memory.

Arguments / parameters

```
<none>
```

Returns

```
<none>
```

Description

Always call this function on the application shutdown!

After calling of this function the SCardX Easy becomes unusable and ready for closing.

C++ Builder syntax:

```
SCardX_Easy->Finalize();
```

5.2.5 GetCardATR

Returns the ATR string of the opened smart card.

Arguments / parameters

Argument Name	Data Type	Description
ReaderName (input)	C++ : BSTR Basic : As String Delphi : WideString	smart card reader name;

All arguments are passed by reference.

Returns

The function returns the ATR string in a hexadecimal format.

Returning value data type
C++ : BSTR Basic : As String Delphi : WideString

C++ Builder syntax:

```
WideString ReaderName="SCM Microsystems Inc. CHIPDRIVE Serial 0";

AnsiString ATR=SCardX_Easy->GetCardATR(&ReaderName);
```

5.2.6 GetCardInfo

Returns the information about the opened smart card.

Arguments / parameters

Argument Name	Data Type	Description
ReaderName (input)	C++ : BSTR Basic : As String Delphi : WideString	smart card reader name;

All arguments are passed by reference.

Returns

The function returns the info string list.

Returning value data type

```
C++     : BSTR
Basic   : As String
Delphi  : WideString
```

Description

This function returns the list of the strings which are divided by the line breaks symbols `#13#10`.

Each info line is formatted as a standart INI file like of this example:

```
[ICC STATE]
ATR STRING=3B 79 94 00 00 59 01 01 0F 01 00
ICC PRESENCE=2
ICC INTERFACE STATUS=255
ICC TYPE PER ATR=1
```

C++ Builder syntax:

```
WideString ReaderName="SCM Microsystems Inc. CHIPDRIVE Serial 0";

AnsiString CardInfo=SCardX_Easy->GetCardInfo(&ReaderName);
```

5.2.7 GetCardInfoFmt

Returns the formatted information about the opened smart card.

Arguments / parameters

Argument Name	Data Type	Description
ReaderName (input)	C++ : BSTR Basic : As String Delphi : WideString	smart card reader name;

All arguments are passed by reference.

Returns

The function returns the info string list.

Returning value data type

```
C++     : BSTR
Basic   : As String
Delphi  : WideString
```

Description

This function returns the list of the strings which are divided by the line breaks symbols `#13#10`.

Each info line is formatted and already prepared for displaying like of this example:

```
ICC STATE
  ATR STRING ...................      3B 79 94 00 00 59 01 01 0F 01 00 01 04 A9
  ICC PRESENCE .................      2
  ICC INTERFACE STATUS .........      255
  ICC TYPE PER ATR .............      1
  CURRENT IO STATE .............      < no info >
```

C++ Builder syntax:

```
WideString ReaderName="SCM Microsystems Inc. CHIPDRIVE Serial 0";

AnsiString CardInfo=SCardX_Easy->GetCardInfoFmt(&ReaderName);
```

5.2.8 GetEventsHistory

Returns the events history strings list from the "Events History" page.

Arguments / parameters

```
<none>
```

Returns

The function returns the events history string list.

```
Returning value data type

C++    : BSTR
Basic  : As String
Delphi : WideString
```

Description

This function returns the list of the events messages from the "Events History" page which are divided by the line breaks symbols `#13#10`:

```
N       Source Event   Value   Event Time
1       MS Smart Card service  Driver loaded         00:02:18  01-XXX-05
2       MS Smart Card service  Service connected             00:02:18  01-XXX-05
3       AKS ifdh 0     Reader state changed   0x00000012 : There is not card in the
reader  00:02:18  01-XXX-05
4       AKS ifdh 1     Reader state changed   0x00000012 : There is not card in the
reader  00:02:18  01-XXX-05
5       SCM Microsystems Inc. CHIPDRIVE Serial 0     Reader state changed   0x001E0012 :
There is not card in the reader       00:02:18  01-XXX-05
```

All fields in the each string are divided by the Tab character `#9`.

This function may be useful for the errors localization during debugging of the remote application.

C++ Builder syntax:

```
AnsiString EventsHistory=SCardX_Easy->GetEventsHistory();
```

5.2.9 GetReaderInfo

Returns the information about the reader.

Arguments / parameters

Argument Name	Data Type	Description
ReaderName (input)	C++ : BSTR Basic : As String Delphi : WideString	smart card reader name;

All arguments are passed by reference.

Returns

The function returns the info string list.

Returning value data type
C++ : BSTR Basic : As String Delphi : WideString

Description

This function returns the list of the strings which are divided by the line breaks symbols `#13#10`.

Each info line is formatted as a standart INI file like of this example:

```
[VENDOR INFO]
VENDOR NAME=SCM Microsystems Inc.
VENDOR IFD TYPE=CHIPDRIVE Serial
VENDOR IFD VERSION=< no info >
VENDOR IFD SERIAL NO=12639860
```

C++ Builder syntax:

```
WideString ReaderName="SCM Microsystems Inc. CHIPDRIVE Serial 0";

AnsiString ReaderInfo=SCardX_Easy->GetReaderInfo(&ReaderName);
```

5.2.10 GetReaderInfoFmt

Returns the formatted information about the reader.

Arguments / parameters

Argument Name	Data Type	Description
ReaderName (input)	C++ : BSTR Basic : As String Delphi : WideString	smart card reader name;

All arguments are passed by reference.

Returns

The function returns the info string list.

Returning value data type

```
C++     : BSTR
Basic   : As String
Delphi  : WideString
```

Description

This function returns the list of the strings which are divided by the line breaks symbols `#13#10`.

Each info line is formatted and already prepared for displaying like of this example:

```
VENDOR INFO
  VENDOR NAME ..................      SCM Microsystems Inc.
  VENDOR IFD TYPE ..............      CHIPDRIVE Serial
  VENDOR IFD VERSION ...........      < no info >
  VENDOR IFD SERIAL NO .........      12639860
```

C++ Builder syntax:

```
WideString ReaderName="SCM Microsystems Inc. CHIPDRIVE Serial 0";

AnsiString ReaderInfo=SCardX_Easy->GetReaderInfoFmt(&ReaderName);
```

5.2.11 GetReadersList

Returns the list of the smart card readers' names which are attached to your PC.

Arguments / parameters

```
<none>
```

Returns

The function returns the readers names string list.

```
Returning value data type

C++     : BSTR
Basic   : As String
Delphi  : WideString
```

Description

This function returns the list of the readers names which are divided by the line breaks symbols `#13#10`:

```
AKS ifdh 0
AKS ifdh 1
SCM Microsystems Inc. CHIPDRIVE Serial 0
```

C++ Builder syntax:

```
AnsiString ReadersList=SCardX_Easy->GetReadersList();
```

5.2.12 IsCardReady

Specifies whether the card in the reader is opened.

Arguments / parameters

Argument Name	Data Type	Description
ReaderName (input)	C++ : BSTR Basic : As String Delphi : WideString	smart card reader name;

All arguments are passed by reference.

Returns

The function returns true or false depends to whether the card in the reader is opened.

Returning value data type

```
C++    : bool
Basic  : As Boolean
Delphi : WordBool
```

C++ Builder syntax:

```
WideString ReaderName="SCM Microsystems Inc. CHIPDRIVE Serial 0";

bool Yes=SCardX_Easy->IsCardReady(&ReaderName);
```

5.2.13 IsLocked

Specifies whether the SCardX Easy is locked for smart card service commands.

Arguments / parameters

<none>

Returns

The function returns true or false depends to whether the SCardX Easy is locked.

Returning value data type

```
C++    : bool
Basic  : As Boolean
Delphi : WordBool
```

C++ Builder syntax:

```
bool Yes=SCardX_Easy->IsLocked();
```

5.2.14 LookUpError

Decodes the error string message from its numerical code.

Arguments / parameters

Argument Name	Data Type	Description
ErrorCodeHex (input)	C++ : BSTR Basic : As String Delphi : WideString	the hexadecimal value of an integer error code;

All arguments are passed by reference.

Returns

The function returns the decoded error string.

```
Returning value data type

C++    : BSTR
Basic  : As String
Delphi : WideString
```

Description

You may decode any error value which you need because this function uses the system function of the your PC operation system.

C++ Builder syntax:

```
WideString ErrorCodeHEX="00000001";

AnsiString ss=SCardX_Easy->LookUpError(&ErrorCodeHEX);
```

5.2.15 LookUpReaderState

Decodes the string value of the card reader state from its numerical code.

Arguments / parameters

Argument Name	Data Type	Description
StateCodeHex (input)	C++ : BSTR Basic : As String Delphi : WideString	the hexadecimal value of an integer state code;

All arguments are passed by reference.

Returns

The function returns the decoded reader state string list.

Returning value data type

C++ : BSTR
Basic : As String
Delphi : WideString

Description

This function returns the list of the strings which are divided by the line breaks symbols `#13#10`.

Each state line is formatted as a standart INI file like of this example:

```
0x00000020=There is a card in the reader
0x00000100=The card in the reader is in use by one or more other applications, but may be
connected to in shared mode
```

C++ Builder syntax:

```
WideString ReaderStateHEX="00000122";

AnsiString ss=SCardX_Easy->LookUpReaderState(&ReaderStateHEX);
```

5.2.16 ReopenReader

Reopens the reader.

Arguments / parameters

Argument Name	Data Type	Description
ReaderName (input)	C++ : BSTR Basic : As String Delphi : WideString	smart card reader name;

All arguments are passed by reference.

Returns

<none>

C++ Builder syntax:

```
WideString ReaderName="SCM Microsystems Inc. CHIPDRIVE Serial 0";

SCardX_Easy->ReopenReader(&ReaderName);
```

5.2.17 SendCardAPDU

Sends the command APDU into the opened smart card and returns the card's answer as a response APDU.

Arguments / parameters

Argument Name	Data Type	Description
ReaderName (input)	C++ : BSTR Basic : As String Delphi : WideString	smart card reader name;
Cla (input)	C++ : BSTR Basic : As String Delphi : WideString	the **Class** hex byte of the command APDU;
Ins (input)	C++ : BSTR Basic : As String Delphi : WideString	the **Instruction** hex byte of the command APDU;
P1 (input)	C++ : BSTR Basic : As String Delphi : WideString	the **Parameter 1** hex byte of the command APDU;
P2 (input)	C++ : BSTR Basic : As String Delphi : WideString	the **Parameter 2** hex byte of the command APDU;
P3Lc (input)	C++ : BSTR Basic : As String Delphi : WideString	the **Length** hex byte of the command APDU;
DataIn (input)	C++ : BSTR Basic : As String Delphi : WideString	the **Data** hex buffer of the command APDU;
Le (input)	C++ : BSTR Basic : As String Delphi : WideString	the **Length** hex byte of the command APDU;
SW1SW2 (output)	C++ : BSTR Basic : As String Delphi : WideString	the **Status Word** (status hex bytes 1 and 2) of the response APDU;
DataOut (output)	C++ : BSTR Basic : As String Delphi : WideString	the **Data** hex buffer of the response APDU;

All arguments are passed by reference.

Returns

The function returns the complete response APDU buffer in a hexadecimal format.

Returning value data type

```
C++     : BSTR
Basic   : As String
Delphi  : WideString
```

Description

Use this function for sending the command APDU's into an opened smart card and for receiving of its response APDU's.

C++ Builder syntax:

```
AnsiString ss             = "";
AnsiString SW1SW2         = "";
AnsiString DataOut        = "";

WideString wReaderName    = "SCM Microsystems Inc. CHIPDRIVE Serial 0";
WideString wCla           = "00";
WideString wIns           = "A4";
WideString wP1            = "00";
WideString wP2            = "00";
WideString wP3            = "02";
WideString wLe            = "";
WideString wDataIn        = "3F00";
WideString wSW            = "";
WideString wDataOut       = "";

ss=   SCardX_Easy->SendCardAPDU(
      &wReaderName,
      &wCla,
      &wIns,
      &wP1,
      &wP2,
      &wP3,
      &wLe,
      &wDataIn,
      &wSW,
      &wDataOut);

if ( ss>"")
   {
   SW1SW2=wSW;
   DataOut=wDataOut;
   ShowMessage("Data sent : \nSW1SW2  = " + SW1SW2 + "\nDataOut = " + DataOut);
   }
```

5.2.18 SendCardDATA

Sends an unformatted data buffer into the opened card and returns the unformatted card's answer.

Arguments / parameters

Argument Name	Data Type	Description
ReaderName (input)	C++ : BSTR Basic : As String Delphi : WideString	smart card reader name;
SentDataBuffer (input)	C++ : BSTR Basic : As String Delphi : WideString	an unformatted send data buffer in a hexadecimal format;

All arguments are passed by reference.

Returns

The function returns an unformatted buffer of the card response data in a hexadecimal format.

Returning value data type
C++ : BSTR Basic : As String Delphi : WideString

Description

Use this function for sending an unformatted data into an opened smart card.

C++ Builder syntax:

```
WideString wReaderName    = "SCM Microsystems Inc. CHIPDRIVE Serial 0";
WideString wDataSend      = "00 A4 00 00 02 3F00";
AnsiString sDataReceived  = "";

sDataReceived=SCardX_Easy->SendCardDATA(&wReaderName,&wDataSend);

if ( sDataReceived>"")
   {
   ShowMessage("Received Data : " + sDataReceived);
   }
```

5.2.19 SetPref_PCSC_OnCardDetect

Sets up the card detecting defaults for using of the MS Smart Card service.

Arguments / parameters

Argument Name	Data Type	Description
AutoOpenReader (input)	C++ : bool Basic : As Boolean Delphi : WordBool	determines whether the card will be opened after detection;
PreferredProtocol (input)	C++ : int Basic : As Long Delphi : Integer	determines the preferred protocol which will be used for the card opening;
PreferredSharingMode (input)	C++ : int Basic : As Long Delphi : Integer	determines the reader sharing mode which will be used for the card opening;
CardClosingMode (input)	C++ : int Basic : As Long Delphi : Integer	determines the card closing mode which will be used by the command ReopenReader ;

All arguments are passed by reference.

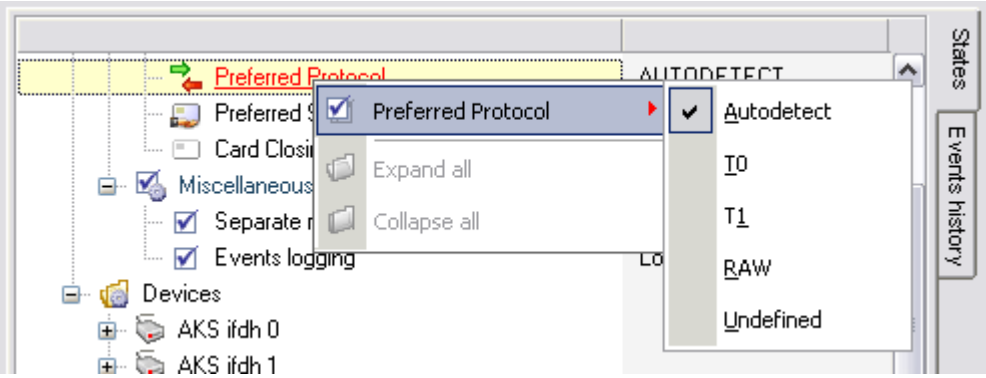
Returns

<none>

Description

Use this command for setting up the card detecting defaults via control's interface.

These preferences' changes becomes visible on the "States" page after calling of this function immediately:



Possible values:

PreferredProtocol

```
xProto_Autodetect      = $00000000
xProto_T0              = $00000001
xProto_T1              = $00000002
xProto_RAW             = $00000003
xProto_Undefined       = $00000004
xProto_Default         = $00000005
```

PreferredSharingMode

```
xSharing_ShareReader          = $00000000
xSharing_ExclusiveUse         = $00000001
xSharing_DirectReaderControl  = $00000002
```

CardClosingMode

```
xClosing_LeaveCard     = $00000000
xClosing_ResetCard     = $00000001
xClosing_UnpowerCard   = $00000002
xClosing_EjectCard     = $00000003
```

C++ Builder syntax:

```
VARIANT_BOOL        AutoOpen              = true;
TxProtocol          PreferredProtocol     = xProto_Autodetect;
TxSharingMode       PreferredSharingMode  = xSharing_ShareReader;
TxCardClosingMode   CardClosingMode       = xClosing_ResetCard;

SCardX_Easy->SetPref_PCSC_OnCardDetect(   &AutoOpen,
                                          &PreferredProtocol,
                                          &PreferredSharingMode,
                                          &CardClosingMode);
```

5.2.20 TrayIconMenuClear

Clears the SCardX Easy tray icon's pop-up menu.

Arguments / parameters

<none>

Returns

<none>

C++ Builder syntax:

```
SCardX_Easy->TrayIconMenuClear();
```

5.2.21 TrayIconMenuCreate

Creates the new pop-up menu of the SCardX Easy's tray icon.

Arguments / parameters

Argument Name	Data Type	Description
MenuItemsList (input)	C++ : BSTR Basic : As String Delphi : WideString	the string list of the new menu items' templates;

All arguments are passed by reference.

Returns

<none>

Description

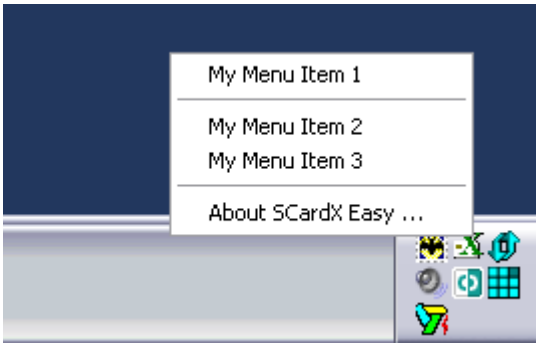
Before calling of this function you need to prepare the menu items' list according to these rules:

- all strings in this list are divided by the line breaks symbols #13#10;
- each new line in the list is the new menu item template;
- each menu item template consists of two parts;
  - the menu item **ID**;
  - the menu item **caption** ;
- these two parts of the menu item template are divided by the "=" character;
- if the menu item template begins with a "-" character the menus divider will be created;

For example your menu items list may be prepared like this one:

```
ID_1=My Menu Item 1
----
ID_2=My Menu Item 2
ID_3=My Menu Item 3
```

These new menu items becomes visible into the tray icon's pop-up menu immediately after calling of this function:



C++ Builder syntax:

```
WideString ww = "ID_1=My Menu Item 1\n----\nID_2=My Menu Item 2\nID_3=My Menu Item 3";

SCardX_Easy->TrayIconMenuCreate(&ww);
```

5.2.22 TrayIconMenuItemSetChecked

Makes the menu item of the tray icon's pop-up menu as checked or unchecked.

Arguments / parameters

Argument Name	Data Type	Description
ItemID (input)	C++ : BSTR Basic : As String Delphi : WideString	the ID string of the menu item which was defined by the TrayIconMenuCreate function;
IsChecked (input)	C++ : bool Basic : As Boolean Delphi : WordBool	the checking flag;

All arguments are passed by reference.

Returns

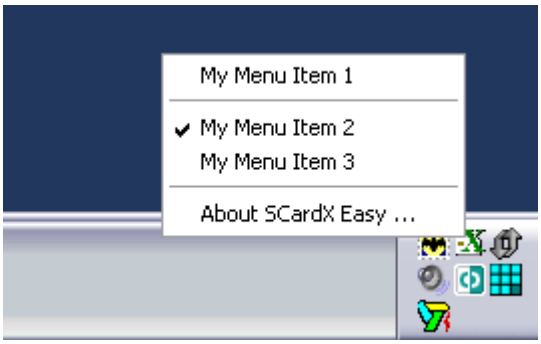
The function returns true if the menu item was found and the command was successful.

Returning value data type

C++ : bool
Basic : As Boolean
Delphi : WordBool

Description

Use this function for marking of the created menu items as checked or unchecked:



C++ Builder syntax:

```
WideString ItemID="ID_1";
VARIANT_BOOL bb=true;

SCardX_Easy->TrayIconMenuItemSetChecked(&ItemID,&bb);
```

5.2.23 TrayIconMenuItemSetDefault

Makes the menu item of the tray icon's pop-up menu as default or standart.

Arguments / parameters

Argument Name	Data Type	Description
ItemID (input)	C++ : BSTR Basic : As String Delphi : WideString	the ID string of the menu item which was defined by the TrayIconMenuCreate function;
IsDefault (input)	C++ : bool Basic : As Boolean Delphi : WordBool	the default item flag;

All arguments are passed by reference.

Returns

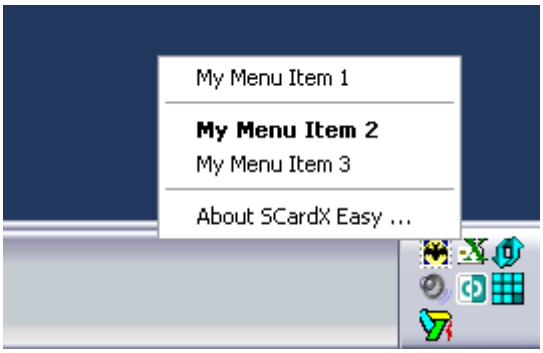
The function returns true if the menu item was found and the command was successful.

Returning value data type

C++ : bool
Basic : As Boolean
Delphi : WordBool

Description

Use this function for marking of the created menu items as default or standart:



C++ Builder syntax:

```
WideString ItemID="ID_1";
VARIANT_BOOL bb=true;

SCardX_Easy->TrayIconMenuItemSetDefault(&ItemID,&bb);
```

5.2.24 TrayIconMenuItemSetEnabled

Makes the menu item of the tray icon's pop-up menu as enabled or disabled.

Arguments / parameters

Argument Name	Data Type	Description
ItemID (input)	C++ : BSTR Basic : As String Delphi : WideString	the ID string of the menu item which was defined by the TrayIconMenuCreate function;
IsEnabled (input)	C++ : bool Basic : As Boolean Delphi : WordBool	the enabling flag;

All arguments are passed by reference.

Returns

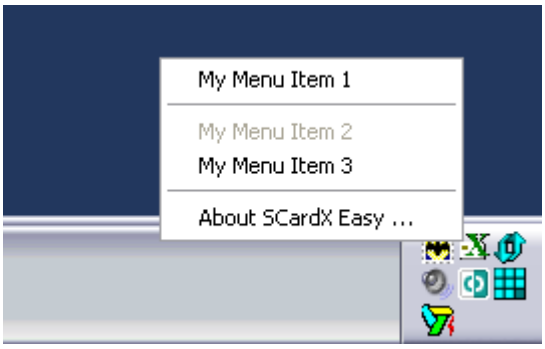
The function returns true if the menu item was found and the command was successful.

Returning value data type

```
C++    : bool
Basic  : As Boolean
Delphi : WordBool
```

Description

Use this function for marking of the created menu items as enabled or disabled:



C++ Builder syntax:

```
WideString ItemID="ID_1";
VARIANT_BOOL bb=false;

SCardX_Easy->TrayIconMenuItemSetEnabled(&ItemID,&bb);
```

5.2.25 Version

Returns the SCardX Easy version string.

Arguments / parameters

<none>

Returns

The function returns the full version string like : Version 1.3

Returning value data type
C++ : BSTR
Basic : As String
Delphi : WideString

C++ Builder syntax:

```
AnsiString Version = SCardX_Easy->Version();
```

5.2.26 VersionMajor

Returns the major digit of the SCardX Easy ActiveX control version.

Arguments / parameters

<none>

Returns

The function returns the integer value of the major digit of the control's version.

Returning value data type
C++ : int
Basic : As Long
Delphi : Integer

C++ Builder syntax:

```
int VersionMajor = SCardX_Easy->VersionMajor();
```

5.2.27 VersionMinor

Returns the minor digit of the SCardX Easy ActiveX control version.

Arguments / parameters

```
<none>
```

Returns

The function returns The integer value of the minor digit of the control's version.

Returning value data type
C++ : int
Basic : As Long
Delphi : Integer

C++ Builder syntax:

```
function VersionMinor: Integer;
```

5.3 Events

User interface events

OnHistoryEvent
OnReaderSelected
OnTrayIconDblClick
OnTrayIconMenuItem

Smart card work events

OnCardDetected
OnCardInvalid
OnCardReady
OnCardWait
OnConnected
OnDataSent
OnDisconnected
OnReadersList
OnReaderStateChanged

Other events

OnERROR
OnLock
OnUnlock

5.3.1 OnCardDetected

Occurs when the card was detected in the reader.

Arguments / parameters

Argument Name	Data Type	Description
ReaderName (output)	C++ : BSTR Basic : As String Delphi : WideString	smart card reader name;

All arguments are passed by reference.

C++ Builder syntax:

```
void __fastcall TMainForm::SCardX_Easy CardDetected(
                    TObject *Sender,
                    BSTR *ReaderName)
{
// ... Your code here ...
}
```

5.3.2 OnCardInvalid

Occurs when the card was detected in the reader but the reader was not able to open it.

Arguments / parameters

Argument Name	Data Type	Description
ReaderName (output)	C++ : BSTR Basic : As String Delphi : WideString	smart card reader name;

All arguments are passed by reference.

C++ Builder syntax:

```
void __fastcall TMainForm::SCardX_Easy CardInvalid(
                    TObject *Sender,
                    BSTR *ReaderName)
{
// ... Your code here ...
}
```

5.3.3 OnCardReady

Occurs when the card was detected and successfully opened in the reader.

Arguments / parameters

Argument Name	Data Type	Description
ReaderName (output)	C++ : BSTR Basic : As String Delphi : WideString	smart card reader name;
ATR (output)	C++ : BSTR Basic : As String Delphi : WideString	the ATR string of an opened card;
ProtocolValue (output)	C++ : int Basic : As Long Delphi : Integer	the real active protocol code of an opened card;
Protocol (output)	C++ : BSTR Basic : As String Delphi : WideString	the real active protocol name of an opened card;

All arguments are passed by reference.

C++ Builder syntax:

```
void __fastcall TMainForm::SCardX_EasyCardReady(
                    TObject *Sender,
                    BSTR *ReaderName,
                    BSTR *ATR,
                    long *ProtocolValue,
                    BSTR *Protocol)
{
// ... Your code here ...
}
```

5.3.4 OnCardWait

Occurs when the card was removed from the reader.

Arguments / parameters

Argument Name	Data Type	Description
ReaderName (output)	C++ : BSTR Basic : As String Delphi : WideString	smart card reader name;

All arguments are passed by reference.

C++ Builder syntax:

```
void __fastcall TMainForm::SCardX_Easy CardWait(
                    TObject *Sender,
                    BSTR *ReaderName)
{
// ... Your code here ...
}
```

5.3.5 OnConnected

Occurs when the smart card service was successfully connected by SCardX Easy.

Arguments / parameters

Argument Name	Data Type	Description
Service (output)	C++ : int Basic : As Long Delphi : Integer	the connected service code;

All arguments are passed by reference.

Possible values:

```
srv_MS_PCSC_SCard_Service   = $00000001
```

C++ Builder syntax:

```
void __fastcall TMainForm::SCardX_EasyConnected(
                    TObject *Sender,
                    long *Service)
{
// ... Your code here ...
}
```

5.3.6 OnDataSent

Occurs when the data was successfully sent into the opened smart card.

Arguments / parameters

Argument Name	Data Type	Description
ReaderName (output)	C++ : BSTR Basic : As String Delphi : WideString	smart card reader name;
SentDataBuffer (output)	C++ : BSTR Basic : As String Delphi : WideString	an unformatted sent data buffer in a hexadecimal format;
ReceivedDataBuffer (output)	C++ : BSTR Basic : As String Delphi : WideString	an unformatted received data buffer in a hexadecimal format;

All arguments are passed by reference.

C++ Builder syntax:

```
void __fastcall TMainForm::SCardX_EasyDataSent(
                    TObject *Sender,
                    BSTR *ReaderName,
                    BSTR *SentDataBuffer,
                    BSTR *ReceivedDataBuffer)
{
// ... Your code here ...
}
```

5.3.7 OnDisconnected

Occurs when the smart card service was disconnected.

Arguments / parameters

<none>

C++ Builder syntax:

```
void __fastcall TMainForm::SCardX_EasyDisconnected(       TObject *Sender)
{
// ... Your code here ...
}
```

5.3.8 OnERROR

Occurs when the error was detected.

Arguments / parameters

Argument Name	Data Type	Description
ErrorSource (output)	C++ : BSTR Basic : As String Delphi : WideString	the source where an error was detected by SCardX Easy;
ErrorCode (output)	C++ : int Basic : As Long Delphi : Integer	the integer error code value;
ErrorString (output)	C++ : BSTR Basic : As String Delphi : WideString	the decoded error string;

All arguments are passed by reference.

C++ Builder syntax:

```
void __fastcall TMainForm::SCardX_EasyERROR(
                    TObject *Sender,
                    BSTR *ErrorSource,
                    long *ErrorCode,
                    BSTR *ErrorString)
{
// ... Your code here ...
}
```

5.3.9 OnHistoryEvent

Occurs when the new event was added into the events grid of the "Events History" page.

Arguments / parameters

Argument Name	Data Type	Description
EventID (output)	C++ : int Basic : As Long Delphi : Integer	the number of the event line;
EventSource (output)	C++ : BSTR Basic : As String Delphi : WideString	the source of the event;
EventBody (output)	C++ : BSTR Basic : As String Delphi : WideString	the event body message;
EventValue (output)	C++ : BSTR Basic : As String Delphi : WideString	the additional event info;
EventTime (output)	C++ : BSTR Basic : As String Delphi : WideString	the event time;

All arguments are passed by reference.

Description

All parameters of this event are equal to the columns values of the events grid of the "Events History" page.

C++ Builder syntax:

```
void __fastcall TMainForm::SCardX_EasyHistoryEvent(
                    TObject *Sender,
                    long *EventID,
                    BSTR *EventSource,
                    BSTR *EventBody,
                    BSTR *EventValue,
                    BSTR *EventTime)
{
// ... Your code here ...
}
```

5.3.10 OnLock

Occurs when the communication data exchange between the SCardX Easy and smart card service is active.

Arguments / parameters

Argument Name	Data Type	Description
Message (output)	C++ : BSTR Basic : As String Delphi : WideString	the string message about the current active operation;

All arguments are passed by reference.

C++ Builder syntax:

```
void __fastcall TMainForm::SCardX_EasyLock(
                    TObject *Sender,
                    BSTR *Message)
{
// ... Your code here ...
}
```

5.3.11 OnReaderSelected

Occurs when the user has selected the reader on the "States" page by mouse clicking on its item.

Arguments / parameters

Argument Name	Data Type	Description
ReaderName (output)	C++ : BSTR Basic : As String Delphi : WideString	smart card reader name;

All arguments are passed by reference.

C++ Builder syntax:

```
void __fastcall TMainForm::SCardX_EasyReaderSelected(
                    TObject *Sender,
                    BSTR *ReaderName)
{
// ... Your code here ...
}
```

5.3.12 OnReadersList

Occurs when the SCardX Easy receives the readers list from the smart card service.

Arguments / parameters

Argument Name	Data Type	Description
ReadersList (output)	C++ : BSTR Basic : As String Delphi : WideString	the list of the readers names which are divided by the line breaks symbols #13#10;

All arguments are passed by reference.

C++ Builder syntax:

```
void __fastcall TMainForm::SCardX_EasyReadersList(
                    TObject *Sender,
                    BSTR *ReadersList)
{
// ... Your code here ...
}
```

5.3.13 OnReaderStateChanged

Occurs when the reader state was changed.

Arguments / parameters

Argument Name	Data Type	Description
ReaderName (output)	C++ : BSTR Basic : As String Delphi : WideString	smart card reader name;
ReaderState (output)	C++ : BSTR Basic : As String Delphi : WideString	the new reader state integer code;
ReaderStateHex (output)	C++ : int Basic : As Long Delphi : Integer	the new reader state hex code;
ReaderStateLookup (output)	C++ : BSTR Basic : As String Delphi : WideString	the decoded new reader state string list; the strings are divided by the line breaks symbols #13#10;

All arguments are passed by reference.

C++ Builder syntax:

```
void __fastcall TMainForm::SCardX_EasyReaderStateChanged(
                    TObject *Sender,
                    BSTR *ReaderName,
                    long *ReaderState,
                    BSTR *ReaderStateHex,
                    BSTR *ReaderStateLookup)
{
// ... Your code here ...
}
```

5.3.14 OnTrayIconDblClick

Occurs when the user double clicks on the tray icon of the SCardX Easy.

Arguments / parameters

```
<none>
```

C++ Builder syntax:

```
void __fastcall TMainForm::SCardX_EasyTrayIconDblClick(   TObject *Sender)
{
// ... Your code here ...
}
```

5.3.15 OnTrayIconMenuItem

Occurs when the user clicks on the menu item of the tray icon's pop-up menu.

Arguments / parameters

Argument Name	Data Type	Description
ItemID (output)	C++ : BSTR Basic : As String Delphi : WideString	the menu item ID string;
IsChecked (output)	C++ : bool Basic : As Boolean Delphi : WordBool	the item checked flag;
IsEnabled (output)	C++ : bool Basic : As Boolean Delphi : WordBool	the item enabled flag;
IsDefault (output)	C++ : bool Basic : As Boolean Delphi : WordBool	the item default flag;
Caption (output)	C++ : BSTR Basic : As String Delphi : WideString	the item caption;

All arguments are passed by reference.

C++ Builder syntax:

```
void __fastcall TMainForm::SCardX_EasyTrayIconMenuItem(
                    TObject *Sender,
                    BSTR *ItemID,
                    VARIANT_BOOL *IsChecked,
                    VARIANT_BOOL *IsEnabled,
                    VARIANT_BOOL *IsDefault,
                    BSTR *Caption)
{
// ... Your code here ...
}
```

5.3.16 OnUnlock

Occurs when the communication data exchange between the SCardX Easy and smart card service was done and the control becomes ready for a new command.

Arguments / parameters

```
<none>
```

C++ Builder syntax:

```
void __fastcall TMainForm::SCardX_EasyUnlock(      TObject *Sender)
{
// ... Your code here ...
}
```

6 Registration

6.1 Unregistered version limitations

Unregistered version of a SCardX Easy ActiveX control works as a demo version only.

These are the unregistered version limitations:

1. your program can send only from 7 up to 10 commands to a smart card per each SCardX Easy start;
2. the SCardX Easy shows unregistered version's reminders in the following areas:
   - in the License info item of the "States" page;
   - in the hint of the tray icon;
   - in the balloon of the tray icon;
3. you can't to hide the tray icon;
4. you may not contact the SCardX Easy support service;

6.2 Licensing

6.2.1 End-User Licenses

If you don't plan to re-distribute SCardX Easy ActiveX control in this case you may purchase one of our End-User Licenses:

1. End-User Personal License - personal usage by a single user;
2. End-User Site License - unlimited usage at a single company;

Licences Prices
Purchase the Personal License
Purchase the Site License

End-User Personal License

Unlimited personal usage by a single user.

You may create your own applications using SCardX Easy ActiveX control and to use its by yourself unlimited:

- license owner may create and unlimited use his own applications which are based on the SCardX Easy ActiveX control for his own personal tasks only;
- any re-distributions are not allowed;

Registered Users Rights :

After purchasing of the End-User Personal License you will be able:

- to unblock your copy of the SCardX Easy ActiveX control by your own Registration Certificate;
- to upgrade the new versions of the SCardX Easy ActiveX control for only 50% of the base price of the Personal License;
- to contact our support service for any questions about the SCardX Easy ActiveX control functionality or about the smart cards basics;

End-User Site License

Unlimited usage at the single company

By purchasing of this license you grants the SCardX Easy ActiveX control and all smart cards applications which are based on this ActiveX to all your developers and to all your company's staff at once.

For example SCardX Easy ActiveX control may be used by your corporate intranet smart cards oriented web site or by others your corporate smart cards applications:

- anybody may use the applications which are based on the SCardX Easy ActiveX control at the any of computers of a company which is an owner of this license;
- any re-distributions are not allowed;

Registered Users Rights :

After purchasing of the End-User Site License you will be able:

- to unblock your copy of the SCardX Easy ActiveX control by your own Registration Certificate;
- to upgrade the new versions of the SCardX Easy ActiveX control for only 50% of the base price of the Site License;
- to request the custom setup packs of the SCardX Easy ActiveX control like the web installation for free;
- to request the custom builds of the SCardX Easy ActiveX control according to your tasks; it may cost more depending on the requested functionality;
- to contact our support service for any questions about the SCardX Easy ActiveX control functionality or about the smart cards basics;

6.2.2 Developers Licenses

You may unlimited re-distribute SCardX Easy ActiveX control as a part of your own software solutions. In this case you may purchase one of our Developer's Licenses:

1. Base Developer's License - unlimited re-distribution without source codes;
2. Developer's License SC - unlimited re-distribution with source codes included;
3. Developer's License FULL - unlimited re-distribution without copyright limitations;

Licences Prices

Base Developers License

Unlimited re-distribution without source codes

Any developer(s) may create applications using SCardX Easy ActiveX control and the licence owner may sale these applications unlimited without any additional payments to SCardSOFT:

- license owner may create, unlimited use and unlimited distribute the applications which are based on the SCardX Easy ActiveX control;
- re-distribution of SCardX Easy ActiveX control allowed as a part of license owner's software without any additional payments to SCardSOFT;
- all rights on the SCardX Easy ActiveX control are reserved by its author;

Developers License SC

Unlimited re-distribution with source codes included

Any developer(s) may create applications using SCardX Easy ActiveX control and the licence owner may sale these applications unlimited without any additional payments to SCardSOFT:

- license owner may create, unlimited use and unlimited distribute the applications which are based on the SCardX Easy ActiveX control;
- re-distribution of SCardX Easy ActiveX control allowed as a part of license owner's software without any additional payments to SCardSOFT;
- all rights on the SCardX Easy ActiveX control are reserved by its author;

- full source codes of SCardX Easy ActiveX control are included;
- the copyright information of the SCardX Easy ActiveX control must be always included in the license of the software which uses the SCardX Easy ActiveX control;

Developer License FULL

Unlimited re-distribution without copyright limitations

Any developer(s) may create applications using SCardX Easy ActiveX control and the licence owner may sale these applications unlimited without any additional payments to SCardSOFT:

- license owner may create, unlimited use and unlimited distribute the applications which are based on the SCardX Easy ActiveX control;
- re-distribution of SCardX Easy ActiveX control allowed as a part of license owner's software without any additional payments to SCardSOFT;
- all rights on the SCardX Easy ActiveX control are reserved by its author;

- full source codes of SCardX Easy ActiveX control are included except of our shareware security subsystem;
- no copyright limitations are present; the control may be re-distributed without our copyright information visible;

6.2.3 Custom versions

What software you can order?

Additionally to our base solutions you can order the following custom software according to your specific tasks:

- custom versions of the SCardX Easy ActiveX control control;
- custom versions of the Smart Card ToolSet program;
- new smart card ActiveX controls;
- new smart card software;

How much does it cost?

The minimal fee for custom software order is a cost of the Site License. The real cost of your order will be calculated according to the requested functionality.

Please be ready to support us additionally, in the case if it will be necessary, by the following:

- a device(s) which will be used by an ordered software;
- smart cards which will be used by an ordered software;
- a device(s) and cards specification(s);

Terms

Our terms of a software creating are from two weeks up to some month depend on the requested functionality.

How to order?

Please read in details how to order a custom software versions on our web site .

6.3 Registration steps

6.3.1 Step 1 : License Query

Run the program "**SCardX Easy Control Center**" from the start menu and make the following:

- open the "Registration" page;
- select "Step 1 : I want now to create the License Query for receiving the Registration Certificate" and press on the "Go to Step 1 : Create the License Query" button;
- fill up all information fields inside the "License Query Maker" window depending to the type of the License which you need and press on the "Make Query" button;
- Open the "License Query" page; there is the License Query's body text there;
- copy the License Query's text into a new e-mail letter and send it to SCardSOFT via e-mail: sales@scardsoft.com ;

We will send you your own Registration Certificate after receiving of your money and after receiving of your License Query during a one working day.

6.3.2 Step 2 : Purchasing the License

You can purchase the License on-line by your credit card.

Your payment will be processed by the Share-It! (Germany) internet payments' service on the highest security level via a secure SSL connection.

Licences Prices
Purchase the License just now

Additionally we accepts the WebMoney and other transfers.

Read more how to purchase the License

We will send you your own Registration Certificate after receiving of your money and after receiving of your License Query during a one working day.

6.3.3 Step 3 : Certificate registration

Copy the text of the Registration Certificate from the received our letter into a memory by "**Copy**" command.

Run the program "**SCardX Easy Control Center**" from the start menu and make the following:

- open the "Registration" page;
- select "Step 3 : I already have my own Certificate and now I want to register the SCardX control" and press on the "Go to Step 3 : Register the SCardX Easy control" button;
- paste the copied text of the received Registration Certificate into an opened "Certificate Registration Form" using the "Paste" button;
- register the program by pressing on the "Register SCardX Easy" button.

Index

- " -

- A -

- B -

- C -

- D -

- E -

- F -

- G -

- I -

- L -

- O -

- P -

- R -

- S -

- T -

- U -

- V -