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
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

## Part I About

1. About SCardX Easy ActiveX control ......................................................... 4
2. Contacts ................................................................................................ 4

## Part II SCardX Easy ActiveX control overview

1. What is the SCardX Easy? ....................................................................... 5
   - SCardX Easy is an ActiveX ................................................................. 5
   - What SCardX Easy can to add into your application? ...................... 5
   - Smart cards in your applications ...................................................... 5
2. Appearance ............................................................................................ 6
   - States page .......................................................................................... 6
   - Events History page .......................................................................... 8
   - ToolBar panel ...................................................................................... 8
   - StatusBar panel ................................................................................ 9
3. Smart card functionality ......................................................................... 10
   - Smart card service ............................................................................ 10
   - Events ................................................................................................ 10
   - Data sending ...................................................................................... 11
4. Additional tools ..................................................................................... 11
   - LookUp service .................................................................................. 11
   - Data ciphering .................................................................................. 12
   - Tray Icon usage ................................................................................ 12
   - Preferences ........................................................................................ 12

## Part III SCardX Easy first start

1. Registering SCardX Easy ActiveX control on the Borland C++ Builder 6 IDE components palette
2. Your first application and the connection testing .................................... 16

## Part IV Your first application. "Hello, cards World !"

1. Demo application .................................................................................. 21
2. New C++ Builder project ...................................................................... 22
3. Interface functions ............................................................................... 22
4. Events .................................................................................................. 24
5. Preparing the connection controls ......................................................... 27
6. Preparing the opened reader controls .................................................... 29
7. Tray Icon .............................................................................................. 34
8. LookUp service .................................................................................... 39
9. Data ciphering ...................................................................................... 40
Part V SCardX Easy interface specification

1 Properties

- ActivePage
- BorderStyle
- borderWidth
- ConnectionState
- EventsHistoryEnabled
- EventsLogging
- SeparateReceivedBytes
- SmartCardService
- Visible
- VisibleEventsHistory
- VisibleStatusBar
- VisibleToolBar
- VisibleTrayIcon

2 Functions

- DES_DecryptString
- DES_EncryptString
- EventsHistoryClear
- Finalize
- GetCardATR
- GetCardInfo
- GetCardInfoFmt
- GetEventsHistory
- GetEventsHistory
- GetReaderInfo
- GetReaderInfoFmt
- GetReadersList
- IsCardReady
- IsLocked
- LookUpError
- LookUpReaderState
- ReopenReader
- SendCardAPDU
- SendCardDATA
- SetPref_PCCS_OnCardDetect
- TrayIconMenuClear
- TrayIconMenuCreate
- TrayIconMenuItemSetChecked
- TrayIconMenuItemSetDefault
- TrayIconMenuItemSetEnabled
- Version
- VersionMajor
- VersionMinor

3 Events

- OnCardDetected
- OnCardInvalid
- OnCardReady
### Part VI Registration

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Unregistered version limitations</td>
<td>98</td>
</tr>
<tr>
<td>2 Licensing</td>
<td>98</td>
</tr>
<tr>
<td>End-User Licenses</td>
<td>98</td>
</tr>
<tr>
<td>Developers Licenses</td>
<td>99</td>
</tr>
<tr>
<td>Custom versions</td>
<td>100</td>
</tr>
<tr>
<td>3 Registration steps</td>
<td>101</td>
</tr>
<tr>
<td>Step 1: License Query</td>
<td>101</td>
</tr>
<tr>
<td>Step 2: Purchasing the License</td>
<td>101</td>
</tr>
<tr>
<td>Step 3: Certificate registration</td>
<td>101</td>
</tr>
</tbody>
</table>
1 About

1.1 About SCardX Easy ActiveX control

SCardX Easy
Smart Card ActiveX control
Version 1.3

Copyright © 2005 by SCardSOFT

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 standard 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.

![States page screenshot]

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 occurred.

<table>
<thead>
<tr>
<th>N</th>
<th>Source</th>
<th>Event</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>MS Smart Card service</td>
<td>Service connected</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>AKS ifdh 0</td>
<td>Reader state changed</td>
<td>0x00000012. There</td>
</tr>
<tr>
<td>4</td>
<td>AKS ifdh 1</td>
<td>Reader state changed</td>
<td>0x00000012. There</td>
</tr>
<tr>
<td>5</td>
<td>SCM Microsystems Inc. CHIFDI</td>
<td>Reader state changed</td>
<td>0x00000012. There</td>
</tr>
<tr>
<td>6</td>
<td>SCM Microsystems Inc. CHIFDI</td>
<td>Waiting for card</td>
<td>Insert card into a read</td>
</tr>
<tr>
<td>7</td>
<td>AKS ifdh 0</td>
<td>Waiting for card</td>
<td>Insert card into a read</td>
</tr>
<tr>
<td>8</td>
<td>AKS ifdh 1</td>
<td>Waiting for card</td>
<td>Insert card into a read</td>
</tr>
<tr>
<td>9</td>
<td>SCM Microsystems Inc. CHIFDI</td>
<td>Reader state changed</td>
<td>0x00000022. There</td>
</tr>
<tr>
<td>10</td>
<td>SCM Microsystems Inc. CHIFDI</td>
<td>Card detected</td>
<td>Card was detected in</td>
</tr>
<tr>
<td>11</td>
<td>SCM Microsystems Inc. CHIFDI</td>
<td>Reader state changed</td>
<td>0x00000122. There</td>
</tr>
<tr>
<td>12</td>
<td>SCM Microsystems Inc. CHIFDI</td>
<td>Card ready</td>
<td>ATR = 38 79 94 00</td>
</tr>
</tbody>
</table>

**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 occurred;

**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 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".

---

**Copyright © 2005 by SCardSOFT**
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:

```cpp
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 occurred 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 occurred 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 on the states of the connection and your readers' states.

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

For managing 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:

```c++
// 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:

```cpp
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:

```c++
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:

```c++
void __fastcall TMainForm::SCardX_EasyReaderStateChanged(TObject *Sender,
    BSTR *ReaderName, long *ReaderState, BSTR *ReaderStateHex,
    BSTR *ReaderStatus, BSTR *ReaderStatusHex, BSTR *ReaderStatusOption, BSTR *ReaderStatusOptionHex,
    int *ReaderError, BSTR *ReaderInfo)
```
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:

![ActiveX control usage example](Image)

**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:

```cpp
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:

```cpp
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;
  - **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**:

```cpp
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:

```cpp
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:

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

4.6 Preparing the opened reader controls

After receiving 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:

```cpp
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:

```cpp
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);
    }
```
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 :

```cpp
void __fastcall TMainForm::Button7Click(TObject *Sender)
{
  try
  {
    WideString ws=ReadersList->Text;
    AnsiString s="";
    if (CheckBox9->Checked)
```
By clicking on the "Get Card ATR" button the application will receive the ATR string of the opened card:

```pascal
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`:

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

    WideString wReaderName = ReadersList->Text;
    WideString wClaro = Cla->Text;
    WideString wCardId = Ins->Text;
    WideString wClaro = P1->Text;
    WideString wCardId = P2->Text;
    WideString wClaro = P3->Text;
    WideString wClaro = Le->Text;
    WideString wDataIn = DataIn->Text;
    WideString wDataOut = "";

    try
    {
```

[Image of Command APDU sending]
try
{
    ss=
    SCardX_Easy-
    >SendCardAPDU( &wReaderName, &wCls, &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 :

```cpp
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 = "";
    }
    DataReceived->Text=sDataReceived;
    sDataReceived = "";
    DataReceived->Text="";
}
```

![Image of the form with controls and output fields](image-url)
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 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 TrayIcon Menu" button, the SCardX Easy will recreate its tray icon pop-up menu:

```c++
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 the menu item's properties take 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:

```c++
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:

```cpp
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:

```cpp
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:

```cpp
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:

```cpp
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:

```cpp
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 OnTrayIconDbiClick event when user double clicks on the tray icon:

void __fastcall TMainForm::SCardX_EasyTrayIconDbiClick(TObject *Sender)
{
    AddEvent ("OnTrayIconDbiClick","" );
}
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:

![Form with controls](image)

**Error LookUp**

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

```cpp
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:

```cpp
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:

![Image of the SCardX Easy ActiveX control]

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:

```cpp
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","\n    Memo->Lines->Add ("DES Key          : "++kk);
    Memo->Lines->Add ("Text for encrypt : "+ww);
```
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:

```cpp
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.

Additionally 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 (Application Protocol Data Unit).

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:

<table>
<thead>
<tr>
<th>Header</th>
<th>Body</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cla Ins P1 P2</td>
<td>[Lc field] [DataIn field] [Le field]</td>
</tr>
</tbody>
</table>

What is the command APDU content?

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

What is the response APDU content?

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

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataOut field</td>
<td>Data</td>
<td>variable</td>
<td>String of bytes received in the data field of the response</td>
</tr>
<tr>
<td>SW1</td>
<td>Status byte 1</td>
<td>1</td>
<td>Command processing status</td>
</tr>
<tr>
<td>SW2</td>
<td>Status byte 2</td>
<td>1</td>
<td>Command processing qualifier</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>CLASS</th>
<th>INS</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT</td>
<td>‘A0’</td>
<td>A4</td>
<td>‘00’</td>
<td>‘00’</td>
<td>‘02’</td>
</tr>
</tbody>
</table>

Command parameters/data:

<table>
<thead>
<tr>
<th>Byte(s)</th>
<th>Description</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 - 2</td>
<td>File ID</td>
<td>2</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>SW1</th>
<th>SW2</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>90</td>
<td>'00'</td>
<td>- normal ending of the command</td>
</tr>
<tr>
<td>91</td>
<td>'XX'</td>
<td>- normal ending of the command, with extra information from the proactive SIM containing a command for the UE. Length 'XX' of the response data</td>
</tr>
<tr>
<td>9E</td>
<td>'XX'</td>
<td>- length 'XX' of the response data given in case of a SIM data download error</td>
</tr>
<tr>
<td>9F</td>
<td>'XX'</td>
<td>- length 'XX' of the response data</td>
</tr>
</tbody>
</table>
So, according to GSM11.11 our command APDU is:

<table>
<thead>
<tr>
<th>Component</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cla</td>
<td>A0</td>
</tr>
<tr>
<td>Ins</td>
<td>A4</td>
</tr>
<tr>
<td>P1</td>
<td>00</td>
</tr>
<tr>
<td>P2</td>
<td>00</td>
</tr>
<tr>
<td>P3/Lc</td>
<td>02</td>
</tr>
<tr>
<td>DataIn</td>
<td>3F00</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Component</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataOut</td>
<td>&lt;none&gt;</td>
</tr>
<tr>
<td>SW1 SW2</td>
<td>9F XX (where XX is the length of the response data)</td>
</tr>
</tbody>
</table>

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.
Description

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

Type:

<table>
<thead>
<tr>
<th>C++</th>
<th>Basic</th>
<th>Delphi</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>As Long</td>
<td>Integer</td>
</tr>
</tbody>
</table>

Possible values:

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

C++ Builder syntax:

```cpp
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:

<table>
<thead>
<tr>
<th>C++</th>
<th>int</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic</td>
<td>As Long</td>
</tr>
<tr>
<td>Delphi</td>
<td>Integer</td>
</tr>
</tbody>
</table>

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.
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:**

<table>
<thead>
<tr>
<th>C++</th>
<th>int</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic</td>
<td>As Long</td>
</tr>
<tr>
<td>Delphi</td>
<td>Integer</td>
</tr>
</tbody>
</table>

**Possible values:**

- stServiceNotConnected = $00000000
- stServiceConnected = $00000001

**C++ Builder syntax:**

```cpp
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:**

<table>
<thead>
<tr>
<th>C++</th>
<th>bool</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic</td>
<td>As Boolean</td>
</tr>
<tr>
<td>Delphi</td>
<td>WordBool</td>
</tr>
</tbody>
</table>

**C++ Builder syntax:**

```cpp
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:**

<table>
<thead>
<tr>
<th>C++</th>
<th>int</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic</td>
<td>As Long</td>
</tr>
<tr>
<td>Delphi</td>
<td>Integer</td>
</tr>
</tbody>
</table>

**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:**

<table>
<thead>
<tr>
<th>C++</th>
<th>bool</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic</td>
<td>As Boolean</td>
</tr>
<tr>
<td>Delphi</td>
<td>WordBoolean</td>
</tr>
</tbody>
</table>

**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:**

<table>
<thead>
<tr>
<th>C++</th>
<th>int</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic</td>
<td>As Long</td>
</tr>
<tr>
<td>Delphi</td>
<td>Integer</td>
</tr>
</tbody>
</table>

**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:**

<table>
<thead>
<tr>
<th>C++</th>
<th>bool</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic</td>
<td>As Boolean</td>
</tr>
<tr>
<td>Delphi</td>
<td>WordBool</td>
</tr>
</tbody>
</table>

**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.

**True**

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

**False**

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:**

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

5.1.12 VisibleToolBar

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

**True**

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

**False**

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

**Type:**

<table>
<thead>
<tr>
<th>Language</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>C++</td>
<td><code>bool</code></td>
</tr>
<tr>
<td>Basic</td>
<td><code>As Boolean</code></td>
</tr>
<tr>
<td>Delphi</td>
<td><code>WordBool</code></td>
</tr>
</tbody>
</table>

**C++ Builder syntax:**

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

### 5.1.13 VisibleTrayIcon

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

**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:**

<table>
<thead>
<tr>
<th>Language</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>C++</td>
<td><code>bool</code></td>
</tr>
<tr>
<td>Basic</td>
<td><code>As Boolean</code></td>
</tr>
<tr>
<td>Delphi</td>
<td><code>WordBool</code></td>
</tr>
</tbody>
</table>

**C++ Builder syntax:**

```cpp
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
### Arguments

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| KeyHEX        | C++: BSTR | the hex data buffer of the DES key value;  
|               | Basic: As String | 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;  
|               | Delphi: WideString | do not use ASCII symbols for the key value: always use the hexadecimal format only; |
| EncryptedDataHEX | C++: BSTR | the hex data buffer of the previously encrypted by DES text string;  
|               | Basic: As String |  
|               | Delphi: WideString |  

All arguments are passed by reference.

### Returns

The function returns the decrypted text string.

<table>
<thead>
<tr>
<th>Returning value data type</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>C++: BSTR</td>
<td></td>
</tr>
<tr>
<td>Basic: As String</td>
<td></td>
</tr>
<tr>
<td>Delphi: WideString</td>
<td></td>
</tr>
</tbody>
</table>

#### C++ Builder syntax:

```cpp
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
<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| KeyHEX        | C++: BSTR Basic: As String Delphi: WideString | the hex data buffer of the DES key value;  
- 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;  
- do not use ASCII symbols for the key value: always use the hexadecimal format only; |
| CryptString   | 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.

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

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReaderName</td>
<td>C++ : BSTR</td>
<td>smart card reader name;</td>
</tr>
<tr>
<td></td>
<td>Basic : As String</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Delphi : WideString</td>
<td></td>
</tr>
</tbody>
</table>

All arguments are passed by reference.

Returns

The function returns the ATR string in a hexadecimal format.

C++ Builder syntax:

```cpp
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

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReaderName</td>
<td>C++ : BSTR</td>
<td>smart card reader name;</td>
</tr>
<tr>
<td></td>
<td>Basic : As String</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Delphi : WideString</td>
<td></td>
</tr>
</tbody>
</table>

All arguments are passed by reference.

Returns

The function returns the info string list.
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 standard 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

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| ReaderName    | 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.
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:**

```cpp
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.

<table>
<thead>
<tr>
<th>Returning value data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>C++ : BSTR</td>
</tr>
<tr>
<td>Basic : As String</td>
</tr>
<tr>
<td>Delphi : WideString</td>
</tr>
</tbody>
</table>

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:

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

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:

```cpp
AnsiString EventsHistory=SCardX_Easy->GetEventsHistory();
```
5.2.9 GetReaderInfo

Returns the information about the reader.

Arguments / parameters

<table>
<thead>
<tr>
<th>Argument</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReaderName</td>
<td>C++: BSTR, Basic: As String, Delphi: WideString</td>
<td>smart card reader name;</td>
</tr>
</tbody>
</table>

All arguments are passed by reference.

Returns

The function returns the info string list.

<table>
<thead>
<tr>
<th>Returning value data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>C++: BSTR</td>
</tr>
<tr>
<td>Basic: As String</td>
</tr>
<tr>
<td>Delphi: WideString</td>
</tr>
</tbody>
</table>

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:

```cpp
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

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReaderName</td>
<td>C++ : BSTR, Basic : As String, Delphi : WideString</td>
<td>smart card reader name;</td>
</tr>
</tbody>
</table>

All arguments are passed by reference.

### Returns

The function returns the info string list.

<table>
<thead>
<tr>
<th>Returning value data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>C++ : BSTR, Basic : As String, Delphi : WideString</td>
</tr>
</tbody>
</table>

### 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.

<table>
<thead>
<tr>
<th>Returning value data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>C++                      : BSTR</td>
</tr>
<tr>
<td>Basic                    : As String</td>
</tr>
<tr>
<td>Delphi                   : WideString</td>
</tr>
</tbody>
</table>

**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:**

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

5.2.12 **IsCardReady**

Specifies whether the card in the reader is opened.

**Arguments / parameters**

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReaderName</td>
<td>C++ : BSTR</td>
<td>smart card reader name;</td>
</tr>
<tr>
<td></td>
<td>Basic : As String</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Delphi : WideString</td>
<td></td>
</tr>
</tbody>
</table>

All arguments are passed by reference.

**Returns**

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

C++ Builder syntax:

```cpp
WideString ReaderName="SCM Microsystems Inc. CHIPDRIVE Serial 0";
bool Yes=SCardX_Easy->IsCardReady(&ReaderName);
```

5.2.14 LookUpError

Decodes the error string message from its numerical code.

Arguments / parameters

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ErrorCodeHex</td>
<td>C++ : BSTR</td>
<td>the hexadecimal value of an integer error code;</td>
</tr>
<tr>
<td></td>
<td>Basic : As String</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Delphi : WideString</td>
<td></td>
</tr>
</tbody>
</table>

All arguments are passed by reference.

Returns
The function returns the decoded error string.

<table>
<thead>
<tr>
<th>Returning value data type</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>C++ : BSTR</td>
<td></td>
</tr>
<tr>
<td>Basic : As String</td>
<td></td>
</tr>
<tr>
<td>Delphi : WideString</td>
<td></td>
</tr>
</tbody>
</table>

**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:**

```cpp
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

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StateCodeHex</td>
<td>C++: BSTR, Basic: As String, Delphi: WideString</td>
<td>the hexadecimal value of an integer state code;</td>
</tr>
</tbody>
</table>

All arguments are passed by reference.

Returns

The function returns the decoded reader state string list.

<table>
<thead>
<tr>
<th>Returning value</th>
<th>data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>C++ : BSTR</td>
<td></td>
</tr>
<tr>
<td>Basic : As String</td>
<td></td>
</tr>
<tr>
<td>Delphi : WideString</td>
<td></td>
</tr>
</tbody>
</table>

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 standard INI file like 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:

```cpp
WideString ReaderStateHEX="00000122";
AnsiString ss=SCardX_Easy->LookUpReaderState(&ReaderStateHEX);
```
5.2.16 ReopenReader

Reopens the reader.

Arguments / parameters

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReaderName</td>
<td>C++: BSTR</td>
<td>smart card reader name;</td>
</tr>
<tr>
<td></td>
<td>Basic: As String</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Delphi: WideString</td>
<td></td>
</tr>
</tbody>
</table>

All arguments are passed by reference.

Returns

<none>

C++ Builder syntax:

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

<table>
<thead>
<tr>
<th>Returning value data type</th>
</tr>
</thead>
</table>
| 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:

```cpp
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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReaderName</td>
<td>C++: BSTR</td>
<td>smart card reader name;</td>
</tr>
<tr>
<td></td>
<td>Basic: As String</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Delphi: WideString</td>
<td></td>
</tr>
<tr>
<td>SentDataBuffer</td>
<td>C++: BSTR</td>
<td>an unformatted send data buffer in a hexadecimal format;</td>
</tr>
<tr>
<td></td>
<td>Basic: As String</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Delphi: WideString</td>
<td></td>
</tr>
</tbody>
</table>

All arguments are passed by reference.

Returns

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

C++ Builder syntax:

```cpp
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 Names, Data Types, and Descriptions

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoOpenReader</td>
<td>C++: bool</td>
<td>Determines whether the card will be opened after detection;</td>
</tr>
<tr>
<td></td>
<td>Basic: As Boolean</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Delphi: WordBool</td>
<td></td>
</tr>
<tr>
<td>PreferredProtocol</td>
<td>C++: int</td>
<td>Determines the preferred protocol which will be used for the card opening;</td>
</tr>
<tr>
<td></td>
<td>Basic: As Long</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Delphi: Integer</td>
<td></td>
</tr>
<tr>
<td>PreferredSharingMode</td>
<td>C++: int</td>
<td>Determines the reader sharing mode which will be used for the card opening;</td>
</tr>
<tr>
<td></td>
<td>Basic: As Long</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Delphi: Integer</td>
<td></td>
</tr>
<tr>
<td>CardClosingMode</td>
<td>C++: int</td>
<td>Determines the card closing mode which will be used by the command ReopenReader;</td>
</tr>
<tr>
<td></td>
<td>Basic: As Long</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Delphi: Integer</td>
<td></td>
</tr>
</tbody>
</table>

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 become 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**

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MenuItemsList</td>
<td>C++: BSTR; Basic: As String; Delphi: WideString</td>
<td>the string list of the new menu items' templates; all arguments are passed by reference.</td>
</tr>
</tbody>
</table>

**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:

![Tray Icon Menu Example](image)

**C++ Builder syntax:**

```cpp
WideString ww = "ID_1=My Menu Item 1
----
ID_2=My Menu Item 2
ID_3=My Menu Item 3";
SCardX_Easy->TrayIconMenuCreate(&ww);
```
5.2.22 TrayIconMenuCreateFunction

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

Arguments / parameters

<table>
<thead>
<tr>
<th>Argument</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ItemID</td>
<td>C++: BSTR</td>
<td>the ID string of the menu item which was defined by the TrayIconMenuCreate function;</td>
</tr>
<tr>
<td></td>
<td>Basic: As String</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Delphi: WideString</td>
<td></td>
</tr>
<tr>
<td>IsChecked</td>
<td>C++: bool</td>
<td>the checking flag;</td>
</tr>
<tr>
<td></td>
<td>Basic: As Boolean</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Delphi: WordBool</td>
<td></td>
</tr>
</tbody>
</table>

Returns

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

Description

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

C++ Builder syntax:

```cpp
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

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ItemID</td>
<td>C++ : BSTR, Basic : As String, Delphi : WideString</td>
<td>the ID string of the menu item which was defined by the TrayIconMenuCreate function;</td>
</tr>
<tr>
<td>IsDefault</td>
<td>C++ : bool, Basic : As Boolean, Delphi : WordBool</td>
<td>the default item flag;</td>
</tr>
</tbody>
</table>

All arguments are passed by reference.

Returns

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

C++ Builder syntax:

```cpp
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

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ItemID</td>
<td>C++ : BSTR</td>
<td>the ID string of the menu item which was defined by the TrayIconMenuCreate function;</td>
</tr>
<tr>
<td></td>
<td>Basic : As String</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Delphi : WideString</td>
<td></td>
</tr>
<tr>
<td>IsEnabled</td>
<td>C++ : bool</td>
<td>the enabling flag;</td>
</tr>
<tr>
<td></td>
<td>Basic : As Boolean</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Delphi : WordBool</td>
<td></td>
</tr>
</tbody>
</table>

All arguments are passed by reference.

Returns

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

<table>
<thead>
<tr>
<th>Returning value data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>C++ : bool</td>
</tr>
<tr>
<td>Basic : As Boolean</td>
</tr>
<tr>
<td>Delphi : WordBool</td>
</tr>
</tbody>
</table>

Description

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

C++ Builder syntax:

```cpp
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

<table>
<thead>
<tr>
<th>Returning value data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>C++ : BSTR</td>
</tr>
<tr>
<td>Basic : As String</td>
</tr>
<tr>
<td>Delphi : WideString</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Returning value data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>C++ : int</td>
</tr>
<tr>
<td>Basic : As Long</td>
</tr>
<tr>
<td>Delphi : Integer</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Returning</th>
<th>value</th>
<th>data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>C++</td>
<td>: int</td>
<td></td>
</tr>
<tr>
<td>Basic</td>
<td>: As Long</td>
<td></td>
</tr>
<tr>
<td>Delphi</td>
<td>: Integer</td>
<td></td>
</tr>
</tbody>
</table>

C++ Builder syntax:

```c
function VersionMinor: Integer;
```
5.3 Events

User interface events

OnHistoryEvent
OnReaderSelected
OnTrayIconDbiClick
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

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReaderName</td>
<td>C++ : BSTR</td>
<td>smart card reader name;</td>
</tr>
<tr>
<td></td>
<td>Basic : As String</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Delphi : WideString</td>
<td></td>
</tr>
</tbody>
</table>

All arguments are passed by reference.

C++ Builder syntax:

```c++
void __fastcall TMainForm::SCardX_EasyCardDetected(
    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

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReaderName</td>
<td>C++ : BSTR</td>
<td>smart card reader name;</td>
</tr>
<tr>
<td></td>
<td>Basic : As String</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Delphi : WideString</td>
<td></td>
</tr>
</tbody>
</table>

All arguments are passed by reference.

C++ Builder syntax:

```c++
void __fastcall TMainForm::SCardX_EasyCardInvalid(
    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
### Table of Arguments

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReaderName</td>
<td>C++ : BSTR, Basic : As String, Delphi : WideString</td>
<td>smart card reader name;</td>
</tr>
<tr>
<td>ATR</td>
<td>C++ : BSTR, Basic : As String, Delphi : WideString</td>
<td>the ATR string of an opened card;</td>
</tr>
<tr>
<td>ProtocolValue</td>
<td>C++ : int, Basic : As Long, Delphi : Integer</td>
<td>the real active protocol code of an opened card;</td>
</tr>
<tr>
<td>Protocol</td>
<td>C++ : BSTR, Basic : As String, Delphi : WideString</td>
<td>the real active protocol name of an opened card;</td>
</tr>
</tbody>
</table>

All arguments are passed by reference.

### C++ Builder Syntax:

```cpp
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

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>C++</th>
<th>Basic</th>
<th>Delphi</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReaderName</td>
<td>BSTR</td>
<td>As String</td>
<td>WideString</td>
</tr>
</tbody>
</table>

All arguments are passed by reference.

C++ Builder syntax:

```cpp
void __fastcall TForm::SCardX_EasyCardWait(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

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service</td>
<td>C++: int</td>
<td>the connected service code;</td>
</tr>
<tr>
<td></td>
<td>Basic: As Long</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Delphi: Integer</td>
<td></td>
</tr>
</tbody>
</table>

All arguments are passed by reference.

Possible values:
- srv_MS_PCSC_SCard_Service = $00000001

C++ Builder syntax:

```cpp
void __fastcall TForm1::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**

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReaderName</td>
<td>C++ : BSTR Basic : As String Delphi : WideString</td>
<td>smart card reader name;</td>
</tr>
<tr>
<td></td>
<td>( output )</td>
<td></td>
</tr>
<tr>
<td>SentDataBuffer</td>
<td>C++ : BSTR Basic : As String Delphi : WideString</td>
<td>an unformatted sent data buffer in a hexadecimal format;</td>
</tr>
<tr>
<td></td>
<td>( output )</td>
<td></td>
</tr>
<tr>
<td>ReceivedDataBuffer</td>
<td>C++ : BSTR Basic : As String Delphi : WideString</td>
<td>an unformatted received data buffer in a hexadecimal format;</td>
</tr>
<tr>
<td></td>
<td>( output )</td>
<td></td>
</tr>
</tbody>
</table>

All arguments are passed by reference.

**C++ Builder syntax:**

```cpp
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:**

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

Occurs when the error was detected.

Arguments / parameters

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ErrorSource</td>
<td>C++: BSTR</td>
<td>the source where an error was detected by SCardX Easy;</td>
</tr>
<tr>
<td></td>
<td>Basic: As String</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Delphi: WideString</td>
<td></td>
</tr>
<tr>
<td>ErrorCode</td>
<td>C++: int</td>
<td>the integer error code value;</td>
</tr>
<tr>
<td></td>
<td>Basic: As Long</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Delphi: Integer</td>
<td></td>
</tr>
<tr>
<td>ErrorString</td>
<td>C++: BSTR</td>
<td>the decoded error string;</td>
</tr>
<tr>
<td></td>
<td>Basic: As String</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Delphi: WideString</td>
<td></td>
</tr>
</tbody>
</table>

All arguments are passed by reference.

C++ Builder syntax:

```c++
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** | **C++**: int, **Basic**: As Long, **Delphi**: Integer | the number of the event line; (output)
**EventSource** | **C++**: BSTR, **Basic**: As String, **Delphi**: WideString | the source of the event; (output)
**EventBody** | **C++**: BSTR, **Basic**: As String, **Delphi**: WideString | the event body message; (output)
**EventValue** | **C++**: BSTR, **Basic**: As String, **Delphi**: WideString | the additional event info; (output)
**EventTime** | **C++**: BSTR, **Basic**: As String, **Delphi**: WideString | the event time; (output)

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:**

```cpp
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                                         |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Message</td>
<td>C++ : BSTR</td>
<td>the string message about the current active operation;</td>
</tr>
<tr>
<td></td>
<td>Basic : As String</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Delphi : WideString</td>
<td></td>
</tr>
</tbody>
</table>

All arguments are passed by reference.

C++ Builder syntax:

```c++
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

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReaderName</td>
<td>C++ : BSTR</td>
<td>smart card reader name;</td>
</tr>
<tr>
<td></td>
<td>Basic : As String</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Delphi : WideString</td>
<td></td>
</tr>
</tbody>
</table>

All arguments are passed by reference.

C++ Builder syntax:

```c++
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
### ReadersList

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReadersList</td>
<td>C++ : BSTR</td>
<td>the list of the readers names which are divided by the line breaks symbols #13#10;</td>
</tr>
<tr>
<td></td>
<td>Basic : As String</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Delphi : WideString</td>
<td></td>
</tr>
</tbody>
</table>

All arguments are passed by reference.

**C++ Builder syntax:**

```cpp
void __fastcall TForm1::SCardX_EasyReadersList(
    TObject *Sender,
    BSTR *ReadersList)
{
    // ... Your code here ...
}
```
5.3.13 OnReaderStateChanged

Occurs when the reader state was changed.

Arguments / parameters

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

All arguments are passed by reference.

C++ Builder syntax:

```cpp
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:**

```c
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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ItemID</td>
<td></td>
<td>C++ : BSTR</td>
<td>the menu item ID string;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Basic : As String</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Delphi : WideString</td>
<td></td>
</tr>
<tr>
<td>IsChecked</td>
<td></td>
<td>C++ : bool</td>
<td>the item checked flag;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Basic : As Boolean</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Delphi : WordBool</td>
<td></td>
</tr>
<tr>
<td>IsEnabled</td>
<td></td>
<td>C++ : bool</td>
<td>the item enabled flag;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Basic : As Boolean</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Delphi : WordBool</td>
<td></td>
</tr>
<tr>
<td>IsDefault</td>
<td></td>
<td>C++ : bool</td>
<td>the item default flag;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Basic : As Boolean</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Delphi : WordBool</td>
<td></td>
</tr>
<tr>
<td>Caption</td>
<td></td>
<td>C++ : BSTR</td>
<td>the item caption;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Basic : As String</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Delphi : WideString</td>
<td></td>
</tr>
</tbody>
</table>

All arguments are passed by reference.

**C++ Builder syntax:**

```cpp
void __fastcall TMainForm::SCardX_EasyTrayIconMenuItem(t
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:

```cpp
void __fastcall TForm::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

- " " -

"Hello cards World!" 44

- A -

About 4
ActivePage property 48
Adding SCardX Easy to application 22
APDU 44
Application shutdown example 43
Application startup example 43
ATR string receiving 62

- B -

BorderStyle property 50
BorderWidth property 50

- C -

C++ Builder component registration 14
C++ Builder components palette 14
Card Closing Mode example 41
Card detecting defaults example 41
Card detecting defaults setting up 75
Card Info example 29
Card Info receiving 62
Card Info receiving formatted 63
Card state checking 68
Clearing the Events History 60
Command APDU 44
Command APDU sending 72
Command APDU sending example 29
Connecting the service 52
Connection example 27
Connection testing 16
ConnectionState property 52
Contacts 4
Custom Software 100

- D -

Demo Application 21
DES decoding and encoding example 40
DES Decrypting 58
DES Encrypting 59
DES_DecryptString function 58
DES_EncryptString function 59
Disconnection example 27

- E -

Error message 90
Events History receiving 65
Events list 84
Events logging enabling/disabling 52
Events logging mode 53
Events receiving example 24
EventsHistoryClear function 60
EventsHistoryEnabled property 52
EventsLogging property 53
Examples path 21

- F -

Finalize example 43
Finalize function 61
First application: "Hello cards World!" 44
First application: Application shutdown 43
First application: Application startup 43
First application: Card detecting defaults 41
First application: Connection controls 41
First application: Data ciphering 40
First application: Events 24
First application: Interface functions 22
First application: LookUp 39
First application: New Project 22
First application: Opened reader controls 29
First application: Tray Icon 34
First start 16
Functions list 57

- G -

GetCardATR function 62
GetCardInfo function 62
GetCardInfoFmt function 63
GetEventsHistory function 65
GetReaderInfo function 66
GetReaderInfoFmt function 66
GetReadersList function 68
GSM11.11 44
- I -
IsCardReady function 68
IsLocked function 69
ISO-7816 44

- L -
Locked control checking 69
LookUp Error code 69
LookUp error code example 39
LookUp Reader State code 71
LookUp reader state code example 39
LookUpError function 69
LookUpReaderState function 71

- O -
OnCardDetected event 85
OnCardInvalid event 85
OnCardReady event 85
OnCardWait event 87
OnConnected event 88
OnDataSent event 89
OnDisconnected event 89
OnERROR event 90
OnHistoryEvent event 90
OnLock event 91
OnReaderSelected event 92
OnReadersList event 92
OnReaderStateChanged event 94
OnTrayIconDbiClick event 95
OnTrayIconMenuitem event 96
OnUnlock event 97

- P -
Preferred Protocol example 41
Preferred Sharing Mode example 41
Properties list 48

- R -
Reader Info example 29
Reader Info receiving 66
Reader Info receiving formatted 66
Readers list receiving 68
Readers list receiving example 27

- S -
SendCardAPDU function 72
SendCardDATA function 75
SeparateReceivedBytes property 53
Separating the received HEX bytes 53
SetPref_PCSC_OnCardDetect function 75
Show / hide the EventsHistory 55
Show / hide the StatusBar 56
Show / hide the ToolBar 56
Show / hide the Tray Icon 57
Smart card service selecting 54
SmartCardService property 54
Status word 44
SW1SW2 44

- T -
Tray Icon double click event 95
Tray Icon example 34
Tray Icon Menu : Clearing 77
Tray Icon Menu : Creating new 78
Tray Icon Menu Item : checked / unchecked 79
Tray Icon Menu Item : default / standart 80
Tray Icon Menu Item : enabled / disabled 81
Tray Icon Menu Item : events receiving 96
TrayIconMenuClear function 77
TrayIconMenuCreate function 78
TrayIconMenuItemSetChecked function 79
TrayIconMenuItemSetDefault function 80
TrayIconMenuItemSetEnabled function 81

- U -
Unformatted data buffer sending 75
Unformatted data buffers sending example 29
Unregistered version limitations 98
- V -

Version function 82
Version Major digit receiving 82
Version Minor digit receiving 83
Version string receiving 82
VersionMajor function 82
VersionMinor function 83
Visible property 54
VisibleEventsHistory property 55
VisibleStatusBar property 56
VisibleToolBar property 56
VisibleTrayIcon property 57