SCardServer V2.14
Technical Documentation

SmartCard Manager, SCARD Interface, Delphi Component

Release 2001-05-31

Contents

The SCardServer

Overview

There are several different manufacturers of smartcards, terminals, and drivers. There are also many industry standards for card protocols. Our goal is to make the integration of smartcards and terminals into your application as easy as possible.
The SCardServer provides the following functionality:

Management of connected terminals

- Plug&Play support
- Management of a selection list for all connected smartcard terminals, similar to the selection lists for printers (e.g. "**`CHIPDRIVE extern at COM1`**")
- Status information on each terminal: Status of the smartcard, serial number and terminal information.
- The most recent configuration data is stored in an INI-file (e.g. COM port assignments)

Management of connected applications

- Management of a of applications currently bound to the SCardServer.
- The SCardServer allows access to the terminal by only one application at a time. When the application is finished with card access, the SCardServer passes control to the next application.
- By registering your application with a card type, the SCardServer can be configured to automatically start your application when that card type is inserted. This can be done by application type (e.g. GSM or EC-card) or by using the AID of the card as an identifier.

Management of memory smartcards

- Automatic detection of the semiconductor type and various parameters including necessity of PINs, write protection and even the page sizes for I²C cards
- Automatic detection of card application data on the card
- Data access with a uniform command set, independent of the card type (e.g. **`Card,MemWrite`** or **`Card,ISOAPDU`**)
- Immediate read access to TLV data fields (Tag Length Value encoding)
- Caches for write- and read access for maximum performance
- PIN management
- More than 50 semiconductor types are currently supported, the most recent list is available at our homepage http://www.towitoko.com

Management of processor smartcards

- Automatic detection of the card type and evaluation of the ATR
- Support of sending commands in transparent mode (1:1 to card without any protocol overhead)
- T0 and T1 are completely implemented according ISO7816-3 including error handling, chaining and all S-blocks
- T0 and T1 protocol parameters are preset according to the ATR
- support of APDU alternately according ISO7816-4, GMS11.11 or CT-API

Access to data of standard card applications

- German health insurance card (“Krankenversichertenkarte“, see command **`Apps,KVK`**)
- German telephone prepaid debit card (“Telefonwertkarte“, see command **`Apps,TWK`**)

Interfaces

The SCardServer runs as a separate background task under Windows 3.11, 95, 98, ME, NT and 2000. Applications can communicate with the SCardServer using one of the following interfaces.

SCARD Interface - SCARD.DLL, SCARD32.DLL

The **SCARD** interface encapsulates the full SCardServer functionality. The implementation on the client's part is extremely easy. Only one DLL-function call is used for all accesses. Window messages do the event handling for your application. The interface is available in 16 bit and 32 bit version under Windows 3.11, 95, 98, ME, NT and 2000. For **DELPHI 1/2/3/4/5** we have a component available, which simplifies the implementation even more. All events are implemented and various lists (terminals, applications, terminal status information, card status information) are available in the form of string lists.

PC/SC Interface

This interface was created by the PC/SC Workgroup (http://www.pcscworkgroup.com) implementations are also available as well for Windows and Linux/Unix.
The use of PC/SC in Windows 95/98/ME and NT requires the installation of the **PC/SC Base Components**. In Windows 2000 the latest version is already included. Windows 3.11 is not supported. A detailed description about Microsoft's implementation of the PC/SC interface can be found in the **Microsoft Windows SDK**. Additional information and a mailing list for developers are available on the internet at http://www.microsoft.com/smartcard. The base components are also available on this page and on the Windows 98 second edition CDROM.
Towitoko provides a **Unit for Delphi 2/3/4/5** which encapsulates most of the PC/SC functions, making this interface available for Delphi applications.
The **MUSCLE** project (Movement for Using Smart Cards in a Linux Environment) created a PC/SC implementation for **Linux/Unix**. Additional information and Linux driver for the CHIPDRIVE smartcard reader are available at their web site http://www.linuxnet.com.

Please be aware that Towitoko does not offer any technical support for this interface.

CT-API Interface - CTAPIW16.DLL, CTAPIW32.DLL

The **CT-API** interface is compatible with CT-API V1.1 (Issued by: Deutsche Telekom AG / PZ Telesec, GMD Forschungszentrum Informationstechnik GmbH, TÜV Informationstechnik GmbH and TeleTrust Deutschland e.V.) and available in 16 bit and 32 bit version under Windows 3.11, 95, 98, ME, NT and 2000. More details on this specification can be found on the internet at http://www.tuevit.de.
The command set is implemented according to the MKT (Multifunktionale Kartenterminals für das Gesundheitswesen, Issuer: GMD Arbeitsgemeinschaft "Karten im Gesundheitswesen").

This interface only gives access to a small fraction of the SCardServer's functionality.

OCF Interface - GEN_TWK.DLL

With this interface developed by **IBM**, the CHIPDRIVE can be used in **Java**-based applications. More details about the **Open Card Framework** (OCF) can be found on the internet at http://www.opencard.org.

Please be aware that Towitoko does not offer any technical support for this interface.

TDEV Interface - TDEV.DLL, TDEV32.DLL

The **TDEV** interface exists for compatibility with our earlier driver support interface. We recommend the use of the new SCARD interface in order to have full access to all new features of the SCardServer.
Available in 16 bit and 32 bit versions under Windows 3.11, 95, 98, ME, NT and 2000.

The SCARD Interface

Basics

Ease of implementation was one of the main goals in developing the SCardServer. The SCardServer offers full support of the PC/SC standard (plus more) while keeping it simple for the programmer and allowing you to start programming right away with minimal effort.

To make the implementation of smartcard access as simple as possible, the SCardServer uses the same syntax for every command. The selection of function calls and transmission of parameters is accomplished using a command string. Input and output data are optional. A command string always contains key words and parameters separated by a comma.

Example 1: This command returns the current terminal type, possible return code: **`0`** = "**`OK`**"

Command:	Str("**`Device,Info,Type`**")
DataIn:	nil
DataOut:	Str("**`CHIPDRIVE extern`**")

Example 2: This command writes 21 characters starting at address 16 to a memory card. Possible return codes: **`0`** = "**`OK`**", **`0x4000`**="**`No card present in terminal`**", **`0x1009`**="**`Terminal is locked`**"

Command:	Str("**`Card,MemWrite,16,21`**")
DataIn:	Str("**`Hello SmartCard World`**")
DataOut:	nil

For testing the previous examples you do **not** need to initialize any parameter or execute any other (administrative) commands - **just start!**

In addition, you gain access to a great number of powerful features which will be especially interesting for all professional users.

DLL Function

All calls of this interface are directly passed to the SCardServer. The function call returns only after processing of the command by the SCardServer. Other Windows messages are also regularly processed while the command is being executed. The SCARD interface can be called recursively in up to four levels.

Both DLLs (16 bit: **`SCARD.DLL`**, 32 bit: **`SCARD32.DLL`**) export the following command:

```
Response = SCardComand (Handle,
                        Cmd,     CmdLen,
                        DataIn,  DataInLen,
                        DataOut, DataOutLen
                       );
```

```
LPINT   Handle      /* pointer to a 32 bit signed integer */
LPSTR   Cmd         /* pointer to a zero terminated string */
LPINT   CmdLen      /* pointer to a 32 bit signed integer */
LPSTR   DataIn      /* pointer to an array of byte or a string */
LPINT   DataInLen   /* pointer to a 32 bit signed integer */
LPSTR   DataOut     /* pointer to an array of byte or a string */
LPINT   DataOutLen  /* pointer to a 32 bit signed integer */
INT     Response    /* 32 bit signed integer */
```

`Handle` In case more instances of DLL are required by the application this handle can be used to distinguish between object instances. The value can be set to zero if only a single instance is used. The SCardServer in this case will do the assignment via the thread- / task handle of your application.

`Cmd` SCardServer command (zero terminated string).

`CmdLen` Length of the command string, if the data transfer to the SCardServer is encrypted; if unencrypted transfer is used, this value must be set to zero.

`DataIn` Pointer to the input data.

`DataInLen` Length of the input data.

`DataOut` pointer to buffer for output data.

`DataOutLen` Maximum length for returned data. Is set to the actual length of the returned data.

`Response` Global return code. Is set to zero after a successful command execution.

Sample code for PASCAL / DELPHI (without using the TSmartCard component)

```
function SCardComand ( var Handle: LongInt;
                       Cmd: Pointer;     var CmdLen: LongInt;
                       DataIn: Pointer;  var DataInLen: LongInt;
                       DataOut: Pointer; var DataOutLen: LongInt
                      ): LongInt; stdcall; external "SCARD32.DLL";
```

Important: Under DELPHI 1 (16 bit version) you must use the 16 bit version of the DLL **SCARD.DLL**, furthermore the **stdcall** does not exist here:

```
...): LongInt; external "SCARD";
```

Sample code for dynamic implementation with C

```
typedef DWORD (__stdcall *SCardCmd)(LPDWORD Handle,
                                    LPCSTR Cmd,     LPINT CmdLen,
                                    LPCSTR DataIn,  LPINT DataInLen,
                                    LPCSTR DataOut, LPINT DataOutLen);
(...)
SCardCmd   pSCardCommand   = NULL;
HANDLE     hScardDLL       = LoadLibrary("SCARD32.DLL");
if (hSCardDLL)
  pSCardCommand = (SCardCmd)GetProcAddress(hScardDLL, "SCardComand");
```

Important: - If you are using a 16 bit version, you have to load the 16 bit DLL **SCARD.DLL**:

```
... = LoadLibrary("SCARD.DLL");
```

- There are no LIB files available which are needed for static import, so only a dynamic import is possible.
- String variables are just pointers to a buffer, don't forget to allocate memory for this buffer.

Sample code for Visual Basic 4/5 and Access/VBA

```
Declare Function SCardComand Lib "SCARD32.DLL" (Handle As Long,
                         ByVal Cmd As String,     CmdLen As Long,
                         ByVal DataIn As String,  DataInLen As Long,
                         ByVal DataOut As String, DataOutLen As Long
                        ) As Long
```

Important: string variables are just pointers to a buffer, don't forget to initialize them, e.g. **DataOut = String(255, 0)**.

Card Status

The SCardServer handles the management of the card. For each application the following information on the status of the card and the terminal is available:

- The terminal status is checked to see it is connected and responding properly. In case of a failure, the status is set to **`ERROR`**.
- If no card present in the terminal, the status is set to **`WAIT`**.
- If a card is inserted, the automatic detection is started, i.e. the exact card type (semiconductor type) is determined and consecutively the card is checked for data of known card applications. While the automatic detection is running the status is set to **`DETECT`**. Card access is **not** possible in this state (error code **`0x4000`**, message "**`No card present in terminal`**").
- If the card cannot be read or another detection failure occurs, the status is set to **`INVALID`**.
- Otherwise control of the card is given to **exactly one** application. This application receives the status **`ACTIVE`** while all other application receive the status **`LOCKED`**.
- This remains until:
  a) The card is removed. The status is set to **`WAIT`** again.
  b) A **`Card,Unlock`** command is issued by the active application.
- In case b) the SCardServer passes control on the card to the next application, which again can pass on control to the next application.
- If all applications have released the card with the **`Card,Unlock`** command the status is set to **`VALID`**, i.e. the card is valid but currently not assigned to any application.
- If an application needs to access the card again (e.g. because of a user request) the control needs to be requested by issuing a **`Card,Lock`** command. The status **`ACTIVE`** is assigned for the application which issued the request - all other applications get the status **`LOCKED`**.
- The active application may release the card by issuing a **`Card,Unlock`** and the status for all applications will return to **`VALID`**.

The card status can always be polled with the command **`Card,Info,Status`**.

Windows Messaging

Under Windows it is much better to transmit status changes using windows messages. This reduces the system load because no continuous polling is necessary.

Your application can register (and unregister) any number of application windows for the receipt of SCardServer messages using the commands **`System,AddHWndMsg`** and **`System,DelHWnd`**.

A message is sent to your application in each of the following cases:

- In case of a status change, e.g. **`WAIT`** → **`DETECT`**,
- if control is passed to another application while the status is **`LOCKED`**.

In the following status change **no message** is sent:

- If your application has requested card access by issuing a **`Card,Lock`**, i.e. the status for your application changes from **`VALID`** to **`ACTIVE`**, **no message** is sent to you. For all other applications the status changes from **`VALID`** to **`LOCKED`** and a message **is sent** to them. The reason for this exception is to have the message **`ACTIVE`** sent only on the first activation after card insertion.

The Windows message is sent to the given window handle(s) using the API function **`PostMessage`**. The message ID can be specified by you with the registration of the window (see **`System,AddHWndMsg`**).

The **W-parameter** indicates the message type:

• **`MsgError`**	= decimal **`100`**	for status changes after **`ERROR`**
• **`MsgWait`**	= decimal **`110`**	for status changes after **`WAIT`**
• **`MsgDetect`**	= decimal **`120`**	for status changes after **`DETECT`**
• **`MsgInvalid`**	= decimal **`130`**	for status changes after **`INVALID`**
• **`MsgValid`**	= decimal **`140`**	for status changes after **`VALID`**
• **`MsgActive`**	= decimal **`150`**	for status changes after **`ACTIVE`**
• **`MsgLocked`**	= decimal **`160`**	for status changes after **`LOCKED`** and for every change of the active application
• **`MsgProgress`**	= decimal **`200`**	for progress display during memory card access
• **`MsgDeviceList`**	= decimal **`300`**	indicates changes of the device list
• **`MsgDeviceSearch`**	= decimal **`301`**	progress display during device search
• **`MsgTaskList`**	= decimal **`310`**	indicates changes of the task list
• **`MsgCardInfo`**	= decimal **`320`**	indicates changes of the CardInfo list

The **low order word of the L-parameter** indicates the index of the active terminal within the terminal list (starting with zero). Exception:

- **`MsgDeviceSearch`**: COM-Port which is checked

The **high order word of the L-parameter** is dependent on the message:

• **`MsgLocked`**	index of the active application within the task list (starting with zero)
• **`MsgProgress`**	completion status from 0 to 100 percent
• **`MsgDeviceSearch`**	completion status from 0 to 100 percent, special values: **`254`**: device OK; **`255`**: No device detected

Deactivate Messages

To stop the processing of window messages in the SCardServer you can call the function SCardComand with the parameters **`Cmd = nil`**, **`CmdLen = 0`**, **`DataIn = nil`**, **`DataInLen = 0`**, **`DataOut = nil`** and **`DataOutlen = -1`**. Using **`DataOutlen = -2`** will re-activate the processing.

Sample code for DELPHI:

```
procedure SCardCmdNoYield (Handle: LongInt);
var L,M,N: LongInt;
begin
  L:=0;
  M:=0;
  N:=-1;
  SCardComand(Handle,nil,L,nil,M,nil,N);
end;

procedure SCardCmdDoYield (Handle: LongInt);
var L,M,N: LongInt;
begin
  L:=0;
  M:=0;
  N:=-2;
  SCardComand(Handle,nil,L,nil,M,nil,N);
end;
```

DELPHI Component TSmartCard

With DELPHI the implementation of card access is even easier. The **`TSmartCard`** component does the following jobs:

- Loads the SCARD Library (16 bit: **`SCARD.DLL`**, 32 bit: **`SCARD32.DLL`**) dynamically and imports the **`SCardComand`** function
- Creates a object instance to the SCardServer
- Creates a window handle and registers it for the receipt of SCardServer events
- Introduces a new exception **`ESmartCard`** and forwards error messages

In the following sections, methods, properties and events of the component TSmartCard are briefly introduced. More detailed information is found in the reference section of the SCardServer commands.

Methods

```
function Comand (const Cmd: String;
                 DataIn: Pointer; DataInLen: LongInt;
                 DataOut: Pointer; DataOutMax LongInt): LongInt
```

This method encapsulates the **`SCardComand`** function for communication with the SCardServer. **`Cmd`** contains the command string. **`DataOutMax`** contains the value for the maximum size of the data structure **`DataOut`**. Both pointers can be assigned with **`nil`** if no data is exchanged. The return value contains the number of bytes written to **`DataOut`**. If an error occurs, an **`ESmartCard`** exception is generated.

`function ComandStr (const Cmd, DataIn: String): String;`

Same as **`Command`** but instead of pointers strings are used for data exchange. The return value resembles DataOut.

`procedure ComandList (const Cmd: String; Lines: TStrings);`

Same as **`Command`** but without input parameter (DataIn = **`nil`**). The result in the form of a string list is placed in lines (e.g. used by DeviceList).

Properties

`Active: Boolean`

Causes the component to load the SCARD library and start the SCardServer, otherwise unload the library.

`AutoUnlock: Boolean`

Allows the automatic release of the card (see command **`Card,Unlock`**) after ending the **`OnActiveCard`** event.

`CardInfo: TStringList`

List of status information on the currently inserted smartcard.

`ConfigMaxPort: Integer`

Denominates the maximum number of available COM-Ports in the **`ConfigMenu`**.

`ConfigMenuItem: TMenuItem`
`ConfigPopupMenu: TPopupMenu`

Either **`ConfigMenuItem`** can be assigned to a menu entry or **`ConfigPopupMenu`** to a popup menu. The component automatically adds all necessary entries for the terminal selection and the configuration of the SCardServer. Only one of these two properties can be set.

`DeviceInfo: TStringList`

List of the terminal status information on the currently selected terminal

`DeviceList: TStringList`

List of all available terminals

`Enabled: Boolean`
Locks all event routines. The library will not be loaded or unloaded. If the SCardServer assigns the control on the card to the component, the command **`Card,Unlock`** is issued immediately to pass on control to the next application (independent of property **`AutoUnlock`**).

`Language: TLanguage = (lngCustom, lngEnglish, lngDeutsch)`
`LanguageText: TStringList`
Specifies the current language. If set to **`lngEnglish`** or **`lngDeutsch`** the component will automatically fill the string list **`LanguageText`**. If set to **`lngCustom`**, you can fill the string list manually with messages in any other language. Any setting will only affect the component. To change the SCardServer's language use the command **`System,SetLng`**.

`StatusLabel: TLabel`
Specifies a Label which automatically displays the SCardServer's current status. The status is taken from **`StatusText`**.

`StatusText: String`
Contains a string describing the SCardServer's current status The text is taken from the string list **`LanguageText`**.

`TaskList: TStringList`
List of all applications / tasks bound to the SCardServer.

`Tag: Longint`
`U`nused in the component, available to your application.

Events

`OnCardActive: TCardEvent`
The card was recognized and activated. The card can now be accessed.

`OnCardDetect: TCardEvent`
A card has been inserted in the terminal. The card cannot be accessed yet!

`OnCardInfoChange: TNotifyEvent`
Event for displaying new data in the **`CardInfo`** list

`OnCardInvalid: TCardEvent`
The card recognition has failed / no valid card!

`OnCardLock: TCardLockEvent`
Another application has started to access the card

`OnCardValid: TCardEvent`
All applications are finished with accessing the card (command **`Card,Unlock`**). It is now possible to access the card again

`OnCardWait: TCardEvent`
No card is present in the terminal. The card has been removed from the terminal

`OnDeviceError: TCardEvent`
Terminal access failed. The terminal to PC connection was interrupted.

`OnDeviceListChange: TNotifyEvent`
Event for displaying new Data in the DeviceList

`OnDeviceSearch: TSearchEvent`
Event for displaying progress during search for a terminal (started with command **`Device,SearchComPort`** or at first start of the SCardServer)

`OnProgress: TProgressEvent`
Event for reporting progress on memory card access

`OnTaskListChange: TNotifyEvent`
Event for displaying new data in the TaskList

Usage with multiple applications

Every time a card is handed from one application to another, a card reset will be performed (see command **Card,Reset**) and any acquired access rights will be lost.

Order of activation of applications

The SCardServer determines the order in which the applications are assigned access to the card. The priority is determined by the following criteria in order of the List.
It is determined if:

- an application has been registered for a special card application type (e.g. SIM-Surf for GSM cards).
- a processor card allows a assignment by the registered name (ISO7816-4).
- a memory card matches a registered mask (byte wise comparison of any memory location).
- an application has registered a AID (contained among the history bytes within the ATR of processor cards or within the ATR (TLV encoding) of a memory card.

If several applications have the same ranking or no criteria were matched the tab sequence of the Windows-desktop is used for determining the first application.

Rules for smooth cooperation of multiple applications

The automatic selection of matching applications and especially the passing of control to the next application can be optimized. Observe the following rules:

- Register reliable criteria
- Allow the SCardServer to start up your application on demand
- Do not open modal dialog boxes as long as your application is not the active one. Otherwise it may happen that several modal dialogs are opened simultaneously!
- Do not use the event **DETECT** for opening dialogs or windows. Instead, just add a line of text to a status line, e.g. "**Card being analyzed, please wait**".
- Do not use the event **INVALID** (invalid card) for modal dialogs.
- Issue the **Card,Unlock** function, if you cannot process the card or if you have finished processing.
- Issue the command **Card,Reset** prior to **Card,Unlock** if you want to reset acquired access rights on the card. If available you should use alternate means of resetting the rights since all caches are erased by resetting the card as well.
- Our suggestion for a terminal selection is a windows menu with the following entries:
  - COM 1 ... COM 8
  - separation line
  - automatic terminal detection
  - separation line
  - list of all connected terminals (command **Device,List**).

  By doing so the user will have all choices for:

a) register new terminals	(**Device,SearchComPort,<Port>**)
b) use the automatic terminal selection	(**Device,Select,-1**)
c) select a explicit terminal -	(**Device,Select,<Index>**)

Global Return Codes

An important advantage of the SCardServer is the uniform error handling by using global return codes. The file **`SCARD.ERR`** contains all values with the assigned text messages. Translations are easily possible by adding a new language section according to the INI-format. Below the error codes are listed in **hexadecimal** form:

`0x0000` "**`OK`**"
(Command was successfully executed)

`0x1001` "**`Serial port not available`**"
(The search on the selected COM port was not possible because the port is not available on the Windows system. The port needs to be configured to be properly recognized by the system)

`0x1002` "**`Serial port is used by another application`**"
(The COM-port is used by another application, e.g. a mouse or a modem)

`0x1008` "**`No terminal detected on selected port`**"
(The COM-port configured properly but no terminal was detected, check the connection)

`0x1009` "**`Terminal is locked by X`**"
(At the moment access to the terminal is not possible because another application is accessing a card or has not released the card yet. “X“ will be replaced by the application’s name)

`0x4000` "**`No card present in terminal`**"

`0x4001` "**`Card was removed during access`**"

`0x4002` "**`Invalid card present in terminal`**"

`0x4004` "**`Card ejection failed`**"
(Reserved for future terminals with automatic card ejection.)

`0x1200` "**`Unknown command`**"
(The command string was not recognized)

`0x1201` "**`Command execution not possible with current card`**"
(Not all commands can be used with all cards - especially those for memory and processor cards)

`0x1202` "**`Command execution not possible with this terminal`**"
(Occurs e.g. if a T0 command is sent to a terminal not supporting processor cards)

`0x1203` "**`Invalid command parameter`**"
(e.g. invalid address range for the command **`Card,MemRead`**)

`0x1310` "**`smartcard access failed`**"
(A non recoverable error occurred during access to the smartcard)

`0x1311` "**`PIN error! X trial(s) left`**"
(PIN-Error for memory smartcards. “X“ is replaced by the remaining number of trials)

`0x2000` "**`Server not available`**"
(The SCardServer failed to start)

Command Set SYSTEM

The system area contains all commands for administration and task management.

System,Info

Determines information about the SCardServer and the status of the command execution.

Command: Str("**`System,Info[,<Field>]`**")
DataIn: nil
DataOut: Str("**`<Data1>#13#10 [<Data2>#13#10[...]]`**")

`<Field>` Optional, only data from one of the following fields:

"**`ErrCode`**"	Error code of the last command.
"**`ErrText`**"	Text of the last error message.
"**`Handle`**"	Handle, assigned to the calling object instance.
"**`Lng`**"	Current language for the calling application.
"**`UsedMemHeap`**"	Used Heap by the SCardServer in bytes.
"**`UsedMemTotal`**"	Used memory by the SCardServer in bytes.
"**`VersionCode`**"	Version of the SCardServer (4 digit BCD encoding).
"**`VersionText`**"	Version as string.

`<DataX>` the requested data.

Example 1: **`System,Info`** returns all values, separated by the characters **`CR+LF`** (**`#13#10`**).

Command: Str(" **`System,Info`**")
DataIn: nil
DataOut: Str("

```
Handle=3
Lng=ENGLISH
VersionCode=0214
VersionText=CardServer V2.14.15
ErrCode=4002
ErrText=Invalid Card present in terminal
UsedMemHeap=312092
UsedMemTotal=1048576" )
```

Example 2: If the command string is supplemented by a keyword, only the specified parameter is returned, e.g. only the current language.

Command: Str("**`System,Info,Lng`**")
DataIn: nil
DataOut: Str("**`ENGLISH`**")

System,Comands

Returns a List of all available commands, each separated by the characters **`CR+LF`** (**`#13#10`**). The command tree can be listed recursively by adding more keywords.

Command: Str("**`System,Commands[,<SubSet>]`**")
DataIn: nil
DataOut: Str("**`<Command1>#13#10[<Command2>#13#10[...]]`**")

`<SubSet>` Optional, list only the commands from this subset.
`<CommandX>` The available commands.

Example 1: List the main commands.

```
Command:  Str( "System,Comands" )
DataIn:   nil
DataOut:  Str( " System
                 Linker
                 Device
                 Card
                 Apps" )
```

Example 2: List the commands from **`Apps,TWK`**.

```
Command:  Str( "System,Comands,Apps,TWK" )
DataIn:   nil
DataOut:  Str( " Seriennummer
                 Hersteller
                 Datum
                 Orginalwert
                 Restwert
                 Chipcode
                 ChipHersteller
                 Betreiber" )
```

System,SetLng

Sets the language for the current application. The error messages are read from the file **`SCARD.ERR`**, which can be easily modified / translated.

Command: Str("**`System,SetLng,<LngStr>`**")
DataIn: nil
DataOut: nil

`<LngStr>` Language (= section string in the file **`SCARD.ERR`**)

Example: Switch to German error messages.

```
Command:  Str( "System,SetLng,GERMAN" )
DataIn:   nil
DataOut:  nil
```

System,ConvertErrCode

Returns the error message text for a given global return code.

Command: Str("**`System,ConvertErrCode,<Code>`**")
DataIn: nil
DataOut: Str("**`<Msg`**>")

`<Code>` Error code in hexadecimal form.
`<Msg>` Error message from the file **`SCARD.ERR`** in the language currently set.

Example: The current language is English, get the error message for the hexadecimal error code **`0x4002`**.

Command: Str("**`System,ConvertErrCode,4002`**")
DataIn: nil
DataOut: Str("**`Invalid Card present in terminal`**")

System,Create

The SCardServer creates an instance for every connected application, based on the task handle. It is not necessary to create an object instance, if only one instance is needed. If multiple instances are needed they have to be set up with this command.

Important: The parameter **`Handle`** from the DLL function **`SCardComand`** needs to be set to **`-1`** for calling, therefore this command will not work when using the Delphi component. However, if multiple instances are needed here, creating several TSmartCard objects is much easier.

Example: Command: Str("**`System,Create`**")
DataIn: nil
DataOut: Str("**`Handle=5`**")

System,Destroy

Releases an object instance which was generated with **`System,Create`**. The SCardServer automatically activates this function if the task handle of the application gets invalidated, i.e. the application was closed.

Command: Str("**`System,Destroy,<Handle>`**")
DataIn: nil
DataOut: nil

`<Handle>` Handle to be released.

Example: Release the handle 3.

Command: Str("**`System,Destroy,3`**")
DataIn: nil
DataOut: nil

System,TaskList

Returns the list of applications and related terminals currently connected to the SCardServer. The application names are separated by the characters **CR+LF** (**#13#10**).

Command:	Str("**System,TaskList**")
DataIn:	nil
DataOut:	Str("**<App1>,<Dev1>#13#10[<App2>,<Dev2>#13#10[...]]**")
<AppX>	Name of the application.
<DevX>	Name of the terminal, port and if necessary the index on this port (for details see command **Device,Info,Port**).

Example:	Command:	Str(" **System,TaskList**")
	DataIn:	nil
	DataOut:	Str(" **SCard Test Tool,'CHIPDRIVE extern I' at COM1** **SmartCard Demo,'CHIPDRIVE twin Slot 1' at COM2** **SIM-surf profi, CHIPDRIVE twin Slot 2' at COM2**")

System,TaskTitle

Allows setting an explicit name for your application which occurs in the application list (see command **System,TaskList**). The default is the text from your application's title in its main window.

Command:	Str("**System,TaskTitle,<Title>**")
DataIn:	nil
DataOut:	nil
<Title>	Application's name (spaces are allowed, but no comma or any special characters).

Example: Set the name to “Hello SmartCard World“

Command:	Str("**System,TaskTitle,Hello SmartCard World**")
DataIn:	nil
DataOut:	nil

System,TaskPath

Reserved for internal use.

System,Upgrade / System,OemRegister

These commands are only present for compatibility reasons with older applications. They do not have a function any longer

System,AddHWndMsg

Registers a window handle and a message value for the notification of your application in case of status changes. Up to 8 windows can be registered

Command:	Str("**`System,AddHWndMsg,<HWnd>,<MsgID>`**")
DataIn:	nil
DataOut:	nil

`<HWnd>` Handle of window to receive messages (decimal).

`<MsgID>` Message value for the notification (Message ID, decimal). The value should be greater or equal to **`WM_USER`** (**`=0x400`**) since this range is reserved for application specific messages.

Example: The main window’s hexadecimal handle is **`0x148B4896`** (**`=344672406`** decimal). The SCardServer’s messages should have the ID **`WM_USER+0x500`** (**`=0x900`** equals **`1524`** decimal).

Command:	Str("**`System,AddHWndMsg,344672406,1524`**")
DataIn:	nil
DataOut:	nil

System,DelHWnd

Deletes a window handle from the list.

Command	Str("**`System,DelHWnd,<HWnd>`**")
DataIn:	nil
DataOut:	nil

`<HWnd>` Handle of the window that had received the messages (decimal).

Example: Delete the window with the hexadecimal handle **`0x148B4896`** (**`=344672406`** decimal).

Command:	Str("**`System,DelHWnd,344672406`**")
DataIn	nil
DataOut	nil

System,SetMainHWnd

If the application’s main window does not exist any longer, the SCardServer assumes that the application has been closed and will automatically delete any open handle(s). Usually the SCardServer detects the main window correctly, but it may be possible that a temporary open window is chosen (like help, password etc.). In this case the main window must be set manually with this command.

Command:	Str("**`System,SetMainWnd,<MainHWnd>`**")
DataIn:	nil
DataOut:	nil

`<MainHWnd>` Main window of the application (decimal).

Example: The main window’s hexadecimal handle is **`0x148B4896`** (**`=344672406`** decimal).

Command:	Str("**`System,SetMainWnd,344672406`**")
DataIn:	nil
DataOut:	nil

System,CryptKey

This command activates the encrypted communication with the SCardServer. Command string and DataIn need to be presented in encrypted form after issuing this command. Correspondingly DataOut is returned in encrypted format by the SCardServer. This command is **not** a function for encrypting data nor will the data itself be stored on the card in encrypted form. Only the communication between the SCardServer and the application is encrypted, but de- and encryption is up to the application.

Important: Since the length of data is always a multiple of 8 when using DES please observe the following rules:

1. Command, DataIn and DataOut have a length which is a multiple of 8. if necessary use dummy characters (**not** 0x00).
2. The command needs to be concluded with a zero character before encrypting.
3. DataIn and DataOut are headed by an 16 bit integer which indicates the actual length of the decrypted data.

Command:	Str("**`System,CryptKey,<Type>`**")
DataIn:	**`<KeyID>`**
DataOut	nil
`<Type>`	"**`DES`**" indicates a standard DES algorithm ("**`CDES`**" and "**`NIL`**" are reserved for internal use).
`<KeyID>`	8 byte KeyID, the DES key is not transmitted directly but in encrypted form (for more details see command **`System,GenCryptKey`**).

Example: Start the encrypted communication. The generated KeyID is **`0x2F 0x83 0xFC 0x5C 0x4F 0x0D 0xBE 0x48`**.

Command:	Str("**`System,CryptKey,DES`**")
DataIn:	**`0x2F 0x83 0xFC 0x5C 0x4F 0x0D 0xBE 0x48`**
DataOut	nil

System,GenCryptKey

Of course the encryption only makes sense if the key itself is not transmitted. Therefore it is necessary to generate a KeyID in a secure environment which is used in the final application phase for hiding the actual DES key. The command **`System.CryptKey`** transmits this KeyID to start the encrypted communication.

Command:	Str("**`System,GenCryptKey,<Type>`**")
DataIn:	**`<DES-Key>`**
DataOut	**`<KeyID>`**
`<Type>`	"**`DES`**" indicates a standard DES algorithm ("**`CDES`**" and "**`NIL`**" are reserved for internal use).
`<DES-Key>`	8 byte DES key which is really used to encrypt the data.
<**`KeyID>`**	The KeyID computed by the SCardServer.

Example: Generate the SCardServer's KeyID for the DES Key **`0x4D 0x59 0x4B 0x45 0x59 0x49 0x53 0x42`**.

Command:	Str("**`System,CryptKey,DES`**")
DataIn:	**`0x4D 0x59 0x4B 0x45 0x59 0x49 0x53 0x42`**
DataOut	**`0x2F 0x83 0xFC 0x5C 0x4F 0x0D 0xBE 0x48`**

Command Set LINKER

These commands are reserved for internal use.

Command Set DEVICE

Device,Info

Returns a list of all terminal parameters. The information relates to the terminal currently assigned to the application (see command **`Device,Select`**).

Command:	Str("**`Device,Info[,<Field>]`**")
DataIn:	nil
DataOut:	Str("**`<Data1>#13#10[<Data2>#13#10[...]]`**")

`<Field>` optional, only data from one of the following fields:

- "**`Status`**" indicates the terminal status:
  - "**`error`**" terminal inaccessible.
  - "**`valid`**" terminal ready.
- "**`Port`**" COM-port and index on which the terminal is connected:
  - "**`COM1"`** COM 1
  - "**`COM2-2"`** COM 2, third terminal
- "**`Type`**", "**`ShortName`**" Device type and short name, according to the following list:
  - **`"CHIPDRIVE micro"`**, "**`CDM`**"
  - **`"CHIPDRIVE extern I"`**, "**`CDX"`**
  - **`"CHIPDRIVE extern II"`**, "**`CDD`**"
  - **`"CHIPDRIVE intern"`**, "**`CDI`**"
  - **`"CHIPDRIVE twin Slot 1"`**, "**`CDT1`**"
  - **`"CHIPDRIVE twin Slot 2"`**, "**`CDT2`**"
  - **`"KartenZwerg"`** ,"**`KTZ`**" (OEM version)
  - **`"CardReader"`**, "**`CCR`**" (OEM version)
- "**`Index`**" Index in the terminal list (see command **`Device,List`**).
- "**`Version`**" Hardware revision number.
- "**`Serial","LotNr`**" Lot and serial number (starting with hardware revision 4.3 a ROM mask is used so these devices don't have an unique lot and serial number any longer; in this case the returned values are not related to a explicit terminal).
- "**`Baudrate`**" Current COM-port transmission speed.
- "**`MaxBaudrate`**" Maximum transmission speed for this device (not the card!).
- "**`Led`**" Status display (see command **`Device,SetLed`**)
- "**`Caps`**" Supported types of smartcards (comma separated string):
  - "**`MEM`**" memory smartcards.
  - "**`CPU`**" processor smartcards
- "**`Mode`**" Select mode for this device:
  - ,,"**`AUTO`**" automatically selected by the SCardServer.
  - "**`SELECTED`**" selected explicitly (see **`Device,Select`**).
- "**`MouseDetect`**" Mouse state:
  - "**`1`**" mouse detected.
  - (empty) no mouse detected.
- "**`PowerFail`**" Error counter for power supply failures(see also command **`Device,CheckPowerFail`**).

`<DataX>` The requested data.

Example 1: **Device,Info** returns all values separated by the characters **CR+LF** (**=#13#10**).

Command Str("**Device,Info**")
DataIn: nil
DataOut: Str(" **Status=valid**
Port=COM2
Type=CHIPDRIVE micro
(...)
MouseDetect=1")

Example 2: If the command string is supplemented by a keyword, only the specified parameter is returned, e.g. only the current device type.

Command: Str("**Device,Info,Type**")
DataIn: nil
DataOut: Str("**CHIPDRIVE micro**")

Device,InfoDeviceID

This is quite similar to the command **Device,Info**, but relating to a specified terminal within the terminal list (see command **Device,List**). There is no need to select this terminal, which would not work if it is already occupied by another application.

Command: Str("**Device,InfoDeviceID,<DevID>[,<Field>]**")
DataIn: nil
DataOut: Str("**<Data1>#13#10[<Data2>#13#10[...]]**")

<DevID> Terminal index ("**0**" = first entry).
<Field> Optional, analogous to **<Field>** at command **Device,Info**.
<DataX> Analogous to **<DataX>** at command **Device,Info**.

Example: Command: Str("**Device,InfoDeviceID,1,Port**")
DataIn: nil
DataOut: Str("**COM1**")

Device,InfoDeviceIDCard

This is quite similar to the command **Card,Info**, but relating to a card in a specified terminal within the terminal list (see command **Device,List**). There is no need to select this terminal which would not work if it is already occupied by another application

Command: Str("**Device,InfoDeviceIDCard,<DevID>[,<Field>]**")
DataIn: nil
DataOut: Str("**<Data1>#13#10[<Data2>#13#10[...]]**")

<DevID> Terminal index ("**0**" = first entry).
<Field> Optional, analogous to **<Field>** at command **Card,Info**.
<DataX> Analogous to **<Data>** at command **Card,Info**.

Example: Request card type in terminal 1.

Command: Str("**Device,InfoDeviceIDCard,1,Type**")
DataIn: nil
DataOut: Str("**SLE4428**")

Device,List

Returns a list of all terminals connected to the SCardServer. The entries are separated by the characters **CR+LF (=#10#13**) each.

Command: Str("**Device,List**")
DataIn: nil
DataOut: Str("**<Dev1>#13#10[<Dev2>#13#10[...]]** ")

<DevX> **N**ame and port, index if necessary.

Example: Command: Str("**Device,List**")
DataIn: nil
DataOut: Str(" **'CHIPDRIVE micro' at COM2**
'CHIPDRIVE exten I' at COM2-1
'CHIPDRIVE twin Slot1' at COM3
'CHIPDRIVE twin Slot2' at COM3")

Device,Refresh

Refreshes the device list, but will not search for new devices.

Command: Str("**Device,Refresh**")
DataIn: nil
DataOut: nil

Device,Select

This command can be used to select a specific terminal from the terminal list or to activate the automatic terminal selection.

Command: Str("**Device,Select[,<Device>]**")
DataIn: nil
DataOut: nil

<Device> Optional, one of these:

"**-1**"	Automatic terminal selection.
<Index>	Index in the terminal list (e.g. "**0**" = first entry).
<Typ>	See list at **Device,Info,Type**).
<ShortName>	See list at **Device,Info,ShortName**.
<Port>	Port and number (e.g. "**COM3**" or "**COM2-1**").

Furthermore, the combinations **<Typ><Port>** and **<ShortName><Port>**

If more than one device matches the given criteria (e.g. COM port for CHIPDRIVE twin or short name when several terminals are present) the first matching device in the list is selected. This list is not supposed to be sorted according to the COM ports. The following criteria apply to an automatic selection:

- If no valid or active cards are present the first valid terminal from the list is selected,
- If a valid card is present in any terminal this terminal becomes the active terminal for a application and remains assigned until the card is removed.

Example: Select CHIPDRIVE micro at COM1.

Command: Str("**Device,Select,CHIPDRIVE micro COM1**")
DataIn: nil
DataOut: nil

Device,SearchComPort

Use this command for initiating a search for a terminal device on the indicated COM-Port. Since all devices are Plug&Play capable this command should be used in case of exception only, e.g. if Plug&Play detection fails or after a previous modification of the terminal list with the command **`Device,Remove`**. If a terminal is detected, the SCardServer determines all device specific data such as device type and serial number. Functional devices are stored in the INI file of the SCardServer. On the next start of the SCardServer, previously detected devices are again tested and installed

Command:	Str("**`Device,SearchComPort [,<Port>]`**")
DataIn:	nil
DataOut:	nil
`<Port>`	Optional, number of the COM port on which the terminal is connected; If no parameter is assigned all COM ports not used otherwise are searched.

Example: Search a device on COM1.

Command:	Str("**`Device,SearchComPort,1`**")
DataIn:	nil
DataOut:	nil

Device,Remove

This command will cause the SCardServer to permanently remove a terminal from its list and the serial port will be released. With the command **`Device,SearchComPort`** a terminal can be reconnected again.

Disconnecting a terminal from the serial port will also cause the SCardServer to release the port itself momentarily. However, the SCardServer will check for about 30 seconds to see if the CHIPDRIVE reappears and in this case, will automatically add it to the terminal list again.

Command:	Str("**`Device,Remove,<DevID>`**")
DataIn:	nil
DataOut:	nil
`<DevID>`	Terminal index ("**`0`**" = first entry).

Example: Delete the terminal 1.

Command:	Str("**`Device,Remove,1`**")
DataIn:	nil
DataOut:	nil

Device,SetMode

Reserved for internal use.

Device,CheckPowerFail

This command checks to see if there is enough power available for the card. In case of a lack of power, the communication with the card can be disturbed or even break down. The consequence of wrong or cut off commands caused but such a failure could cause a card to be locked or even permanently damaged. An application should check this counter if several card commands return invalid data or unusual error codes. To recharge the battery of a CHIPDRIVE (if available), connect the device to the running computer for about half an hour.

Command:	Str("**Device,CheckPowerFail**")
DataIn:	nil
DataOut:	Str("**<FailCount>**")
<FailCount>	Power failure counter that is incremented on each error in the card's power supply.

Example:	Command:	Str("**Device,CheckPowerFail**")
	DataIn:	nil
	DataOut:	Str("**0**")

Device,SetLed

This command refers to the active terminal of your application and controls its status display. We do not recommend a manual control of this LED since the SCardServer usually does this. Any LED setting will last until the next LED command is issued. This can either be issued by your application or by the SCardServer in case of a card event or card access. An application can never get permanant control over the status LED.

Command	Str("**Device,SetLed,<ColorStr>**"")
DataIn:	nil
DataOut:	nil
<ColorStr>	Max. 8 characters, according to this color ID: "**0**" = off, "**1**" = red, "**2**" = green, "**3**" = yellow.

Example1:	Slow red blinking.	
	Command:	Str("**Device,SetLed,0011**")
	DataIn:	nil
	DataOut:	nil
Example2:	Steady green signal.	
	Command:	Str("**Device,SetLed,2**")
	DataIn:	nil
	DataOut:	nil
Example3:	Red green yellow cycling.	
	Command:	Str("**Device,SetLed,123**")
	DataIn:	nil
	DataOut:	nil

Command Set CARD

Card,Info

Returns a list with card specific information.

Command: Str("**Card,Info[,<Field>]**")
DataIn: nil
DataOut: Str("**<Data1>#13#10[<Data2>#13#10[...]]**")

<InfoField> optional, only the data from one of these fields:

"**Status**" Card state:
- "**error**" terminal/card error.
- "**wait**" no card in slot.
- "**detect**" card inserted and detection in progress.
- "**invalid**" card invalid or not identified.
- "**valid**" card valid, available to any application.
- "**active**" card valid, locked by your application.
- "**locked**" card valid, locked by another application.

"**LockedBy**" Index and name of the active application within the task list, comma separated string (see command **System,Tasklist**).

"**LinkerApps**" Reserved for internal use.

"**LinkerCards**" Reserved for internal use.

"**PtsAuto**" Reserved for internal use.

"**PtsBinary**" The four characters PTSS, PTS0, PTS01 and PCK of the PTS (PTS2 and PTS3 are not used and skipped), only available after a PTS has been issued.

"**PtsBinaryLen**" See **PtsBinary**.

"**Baudrate**" Current baud rate of the card.

"**CardCount**" Number of cards inserted since the last reboot.

"**CardPower**" Power state of the current card; a memory card is deactivated about two seconds after the last access, a smart card remains active.
- "**0**" card active.
- "**1**" card deactivated.

"**Type**" The chip type of a memory card. The most recent list can be found on our homepage at http://www.towitoko.de.

"**Protocol**" The card’s current protocol:
- "**ATR**" cards with special bit protocols (e.g. SLE4406/4436).
- "**2W**" 2-Wire protocol.
- "**3W**" 3-Wire protocol.
- "**I2C**" I2C-bus protocol.
- "**I2CX**" I2C-bus protocol with 2 byte addressing.
- "**XC...** " special I2C-bus protocols for XICOR chips.
- "**T0**", "**T1**" CPU smart card protocols..

"**Apps**" List of detected card application modules (separated by comma):
- "**KVK**" valid German health insurance card.
- "**TWK**" German prepaid telecom debit card.
- "**TLV**" valid TLV structure.

"**MemSize**" Memory cards only: size of accessible data memory in bytes.

"**PinSize**" Memory cards only: size of the PIN in bytes.

"**PinCnt**" Memory cards only: remaining number of PIN entry trials.

"**PageSize**" I2C memory cards only: page size for write commands.

"**ErrMem**" Memory cards only: error counter for write and verify access.

"**ErrMemPB**" Memory cards only: error counter for write and verify access to the protection bits.
"**AtrBinary**" ATR in binary form (not available for all memory cards).
"**AtrBinarySize**" See **AtrBinary**.
"**AtrHistory**" T0/T1 smartcards only: history bytes according to ISO7813-3.
"**AtrHistorySize**" See **AtrHistory**.
"**TS**", "**T0**",
"**TA1**"-"**TD8**" Decoded ATR according to ISO7816-3.
"**SAD**", "**DAD**" T1 smartcards only: source and destination address.
"**IFSC**", "**IFSD**" T1 smartcards only: buffer size of card and terminal.
"**CWT**", "**BWT**" T0/T1 smartcards only: character and block wait time.

<DataX> The requested data.

Example1: **Card,Info** returns all values, separated by the characters **CR+LF** (**=#13#10**).

Command: Str(" **Card,Info**")
DataIn: nil
DataOut: Str(" **Status=active**
LockedBy=1,Value Card Station
Type=CPU
Protocol=T1
CardCount=4
CardPower=0
Baudrate=9600
(....)")

Example 2: If the command string is supplemented by a keyword, only the specified parameter is returned, e.g. only the current card's historical bytes.

Command: Str("**Card,Info,ATRHistroy**")
DataIn: nil
DataOut: **0x65 0x63 0x06 0x03 0x14 0x02 0x50 0x00 0x06 0x51 0x04 0xB7 0x3E 0x01 0x41**

Card,Lock

Locks a card form access by other applications. The command can only be executed if a valid card is present in the terminal and no other application currently is processing this card (**Card,Info,Status** = "**VALID**"). The command only needs to be issued if a card is to be processed again after it has been released with **Card,Unlock** for other applications.

Command: Str("**Card,Lock**")
DataIn: nil
DataOut: nil

Card,Unlock

This command is used to release a card for processing by other applications. Before assigning the card to another application a reset is performed and any acquired access rights are lost.

Command: Str("**Card,Unlock**")
DataIn: nil
DataOut: nil

Card,MemDisableCache / Card,MemEnableCache

Disables or enables the cache function for memory cards, i.e. even data already read is physically read again from the card for each access. The card is enabled by default.

Commands: Str("**`Card,MemDisableCache`**")
Str("**`Card,MemEnableCache`**")

DataIn: nil
DataOut: nil

Card,InitBwtCwt / Card,InitSadDad / Card,InitIfsdIfsc

Allows setting the Block Waiting Time (time-out of the first character of a block in ms) and the Character Waiting Time (time-out for the following characters in ms) manually. The initial values that are taken from the ATR are overwritten.

T=1 smartcards also allow setting a Source Address and a Destination Address. Initially both values are set to zero. Furthermore, the terminal's buffer size and the smartcard's buffer size can also be set. The initial values are taken from the ATR.

Commands: Str("**`Card,InitBwtCwt,<Bwt>,<Cwt>`**")
Str("**`Card,InitSadDad,<Sad>,<Dad>`**")
Str("**`Card,InitIfsdIfsc,<Ifsd>,<Ifsc>`**")

DataIn: nil
DataOut: nil

`<Bwt>`	Block Waiting Time (decimal **`0`** – **`60.000`**, i. e. 1 ms to 60 seconds)
`<Cwt>`	Character Waiting Time (decimal **`0`** – **`60.000`**, i. e. 1 ms to 60 seconds)
`<Sad>`	Source Address (decimal **`0`** - **`255`**).
`<Dad>`	Destination Address (decimal **`0`** - **`255`**).
`<Ifsd>`	Terminal 's buffer size (decimal **`0`** - **`255`**).
`<Ifsc>`	Smartcard 's buffer size (decimal **`0`** - **`255`**).

Example: Set a block waiting time of 1600 ms and a character waiting time of 4 ms.

Command: Str("**`Card,InitBwtCwt,1600,4`**")
DataIn: nil
DataOut: nil

Card,Reset

This command initiates a hardware reset of the card and any obtained access rights are lost. Such a reset is also performed every time a card is handed from one application to another.

Command: Str("**`Card,Reset`**")
DataIn: nil
DataOut: nil

Card,APDU

This command sends an APDU to the card and receives the card's response. 'Case 1', 'Case 2 short' up to 'Case 4 short' with maximum data length of 254 bytes are supported. The translation to the T0/T1 protocol is done according to ISO7816-4. GSM return codes (9Fxx, 61xx and 6Cxx) are **not** interpreted, this complies to the CTAPI specification of an APDU and **not** to ISO7816-4.

Command:	Str("**`Card,APDU`**")
DataIn:	**`<CLA><INS><P1><P2>[<LC><DataIn>][<LE>]`**
DataOut:	**`<SW1><SW2>[<DataOut>]`**

`<CLA><INS>`	Class and Instruction, one byte each.
`<P1><P2>`	Parameter 1 and 2, one byte each.
`<LC><DataIn>`	Optional, **`<LC>`** (one byte) specifies the number of byte to be sent to the card, **`<DataIn>`** contains these data bytes.
`<LE>`	optional, one byte, expected length of **`<DataOut>`** in byte.
`<SW1><SW2>`	Status Word, byte 1 and 2.
`<DataOut>`	Optional, if **`<LE>`** is set it specifies the length in bytes.

The following cases are supported. Maximum data length is 254 byte.

- ISO CASE 1: Command without data.

  DataIn: **`<CLA><INS><P1><P2>`**
  DataOut: **`<SW1><SW2>`**

  Example: DataIn: **`0x00 0x42 0x05 0x01`**
  DataOut: **`0x90 0x00`**

- ISO CASE 2 short: Command with response data from the card (**`<Le>`**: **`0x00`** - **`0xFF`**).

  DataIn: **`<CLA><INS><P1><P2><LE>`**
  DataOut: **`<SW1><SW2><DataOut>`**

  Example: DataIn: **`0x00 0x42 0x05 0x02 0x03`**
  DataOut: **`0x90 0x00 0x54 0x57 0x4B`**

- ISO CASE 3 short: Command with data block for the card (**`<Lc>`**: **`0x01`** - **`0xFF`**).

  DataIn: **`<CLA><INS><P1><P2><LC><DataIn>`**
  DataOut: **`<SW1><SW2>`**

  Example: DataIn: **`0x00 0x42 0x05 0x03 0x03 0x54 0x57 0x4B`**
  DataOut: **`0x90 0x00`**

- ISO CASE 4 short: Command with data block **and** response data (**`<Le>`**: **`0x01`** - **`0xFF`**).

  DataIn: **`<CLA><INS><P1><P2><LC><DataIn><LE>`**
  DataOut: **`<SW1><SW2><DataOut>`**

  Example: DataIn: **`0x00 0x42 0x05 0x04 0x03 0x54 0x57 0x4B 00x4`**
  DataOut: **`0x90 0x00 0x4A 0x55 0x50 0x21`**

Card,ISOAPDU

This Command is similar to **`Card,APDU`**, but the return codes 61xx and 6Cxx are interpreted and lead to a GetResponse command, i.e. T0- and T1-cards react identical on APDU level. This complies with the exact ISO 7816-4 requirements and allows a T0 / T1 independent APDU.

Important: GSM-cards operate with the T0-protocol but are (unfortunately) not compatible with ISO-standards with respect to the APDU since the return code 9Fxx is used instead of 61xx.

Card,ISOAPDUEXTT0 / Card,ISOAPDUEXTT1

This command works similar to **`Card,APDU`**, but it allows sending extended APDUs with more than 256 bytes of data to a T=0 or T=1 smartcard. Currently, only few smartcards support this feature. Details about the structure of extended APDUs can be found in ISO 7816-4.

Card,T1

This command executes a T1 command (including chaining if necessary). The same ADPU cases mentioned in **`Card,APDU`** are supported. But in contrast to **`Card,APDU`**, all data will be passed to the card transparently. This can be necessary if crypted APDUs are used.

Command:	Str("**`Card,T1`**")
DataIn:	**`<DataIn>`**
DataOut:	**`<SW1><SW2>[<DataOut>]`**

`<DataIn>`	Can be an APDU or raw (crypted) data
`<SW1><SW2>`	Status word, byte 1 and 2.
`<DataOut>`	Optional, response data depending on **`<DataIn>`**.

Example: analogous to **`Card,APDU`**

Card,T0TX

This command sends a T0 command with data to the card:

Command:	Str("**`Card,T0TX`**")
DataIn:	**`<CLA><INS><P1><P2><P3><DataIn>`**
DataOut:	**`<SW1><SW2>`**

`<CLA><INS>`	Class and Instruction, one byte each.
`<P1><P2><P3>`	Parameter 1, 2 and 3, one byte each.
`<DataOut>`	Data block.
`<SW1><SW2>`	Status Word, byte 1 and 2.

Example: analogous to **`Card,APDU`**

Card,T0RX

This command sends a T0 command to the card and receives data from the card:

Command:	Str("**Card,T0RX**")
DataIn:	**<CLA><INS><P1><P2><P3>**
DataOut:	**<SW1><SW2><DataOut>**

<CLA><INS>	Class and Instruction, one byte each.
<P1><P2><P3>	Parameter 1, 2 and 3, one byte each.
<SW1><SW2>	Status word, byte 1 and 2.
<DataOut>	Response data from the card, length depends on the command.

Example: analogous to **Card,APDU**

Card,TspTxRxLen

Sends a string to the card and receives a given number of bytes from the card. The command does not respect any protocols, i.e. sends and receives absolutely transparent on byte level. The time-out values CWT and BWT are effective here as well.

Command:	Str("**Card,TspRxLen,<RxLen**")
DataIn:	**<DataIn>**
DataOut:	**<DataOut>**

<RxLen>	number of bytes expected as response data from the card (decimal).
<DataIn>	data to be sent to the card
<DataOut>	response data, length given in **<RxLen>**

Example: analogous to **Card,APDU**

Card,PTS

Sets the smart card's protocol and data transfer speed. If these features are supported by the card. The PTS (Protocol Type Selection) consists of the six characters PTSS (PTS-ID), PTS0-PTS3 and PCK (Checksum). The characters PTS2 and PTS3 are currently unused.

Command:	Str("**Card,PTS,<N1><N2><N3>**")
DataIn:	nil
DataOut:	nil

<N1>	Protocol (Bits 0-3 of PTS0, "**0**" = T0 and "**1**" = T1 are valid)
<N2>	Clock rate conversion factor FI (Bits 4-7 of PTS1, "**0**" to "**F**")
<N3>	Baud rate adjustment factor DI (Bits 0-3 of PTS1, "**0**" to "**F**")

N2 and N3 are necessary to change the smart card's transfer rate. The default values are obtained from the ATR's T_{A1} character. Please refer to ISO 7816-3 and the smart card's documentation for other settings for N2 and N3 (FI and DI) and thus for the possible transfer rates.

Example: Set protocol T0, FI = 1, DI = 1

Command:	Str("**Card,PTS,011**")
DataIn:	nil
DataOut:	nil

Card,MemRead

Reads the selected area from a memory card’s data memory. Independent of any of the supported memory chips (most recent list available at http://www.towitoko.de).

Command:	Str("**Card,MemRead,<Offset>,<Len>**")
DataIn:	nil
DataOut:	**<DataOut>**

<Offset> Offset of the first byte to read (0 = first byte of card memory) .
<Len> Number of bytes to read.
<DataOut> Data read.

Example: Read 21 bytes starting at offset 16.

Command:	Str("**Card,MemRead,16,21**")
DataIn:	nil
DataOut:	**0x48 0x65 0x6C 0x6C 0x6F 0x20 0x53 0x6D 0x61 0x72 0x74 0x43 0x61 0x72 0x64 0x20 0x57 0x6F 0x72 0x6C 0x64** (as String: "Hello SmartCard World")

Card,MemWrite

Writes data to a memory card’s data memory, independent of any of the supported memory chips (most recent list available at http://www.towitoko.de). If the cache function is active (default), only data bytes which have actually changed are written to the card – but this only works if the same data areas have been previously read from the card. Every write access is (internally) followed by a verify command. In case of a write error, **Card,MemReadStatus** can be used to retrieve the exact result.

Command:	Str("**Card,MemWrite,<Adr>,<Len>**")
DataIn:	**<DataIn>**
DataOut:	nil

<Adr> Offset of the first bye to write.
<**Len>** Number of bytes to write.
<DataOut> Data to write.

Example: Write "Hello SmartCard World" (21 characters/bytes) at offset 16.

Command:	Str("**Card,MemWrite,16,21**")
DataIn:	**0x48 0x65 0x6C 0x6C 0x6F 0x20 0x53 0x6D 0x61 0x72 0x74 0x43 0x61 0x72 0x64 0x20 0x57 0x6F 0x72 0x6C 0x64** (as string "Hello SmartCard World")
DataOut:	nil

Card,MemVerify

Performs a byte by byte comparison between the transmitted data and the data stored on a memory card. The number of errors in data bytes and write protection bits can also be retrieved with **`Card,Info`**. In case of a verify error the error code **`0x1310`** (”**`Card access failed`**“) is returned. By using **`Card,MemReadStatus`** the exact result of the comparison can be retrieved.

Command: Str("**`Card,MemVerify,<Adr>,<Len>`**")
DataIn: **`<VerifyData>`**
DataOut: nil

`<Adr>` Offset of the first byte to check (0 = first byte in card memory).
<**`Len>`** Number of bytes to compare, must be length of **`<VerifyData>`**.
`<VerifyData>` Data bytes to compare with card memory.

Example: Verify "Hello SmartCard World" (21 characters/bytes) at offset 16.

Command: Str("**`Card,MemVerify,16,21`**")
DataIn: **`0x48 0x65 0x6C 0x6C 0x6F 0x20 0x53 0x6D 0x61 0x72`**
`0x74 0x43 0x61 0x72 0x64 0x20 0x57 0x6F 0x72 0x6C`
`0x64` (as string "Hello SmartCard World")
DataOut: nil

Card,MemReadPB / Card,MemWritePB / Card,MemVerifyPB

These three commands are similar to the previous three commands **`Card,MemRead`** / **`Card,MemWrite`** and **`Card,MemVerify`**, but the functions do not relate to the data memory but instead to the write protect information of the card. Some cards allow the activation of the write protection independently for any (or a subset) of the data memory.

Important: Once a write protection bit is set, some cards (e.g. SLE4428 or SLE4442) don’t allow resetting it again. Thus, this data byte can’t be changed any longer

Every byte transmitted in **`<DataIn>`** or received in **`<DataOut>`** resembles the information on the write protection of one data byte on the card. The following values are defined:

`0x00`: write protection **not** active.
`0x01`: write protection active.

Card,MemSetPB

Activates the write protection for the specified address range of the card. The same result can be archived with **`Card,MemWritePB`** but for most cases this command is easier to use.

Command: Str("**`Card,MemVerify,<Adr>,<Len>`**")
DataIn: nil
DataOut: nil

`<Adr>` Offset of the first byte to set the protection bit for (0 = first byte).
<**`Len>`** Number of bytes to set the protection bits for.

Example: Set the protection bit for the next 21 bytes starting at offset 16.

Command: Str("**`Card,MemSetPB,16,21`**")
DataIn: nil
DataOut: nil

Card,MemReadStatus

Reads status information on the cache, write protection and verify errors.

Command:	Str("**`Card,MemReadStatus,<Adr>,<Len>`**")
DataIn:	nil
DataOut:	**`<Status>`**

`<Adr>` Offset of the first byte to read the status information for (0 = first byte).

<**`Len>`** Number of bytes to read the status information for.

`<Status>` The status information is encoded as follows:

Bit 7 (**`0x80`**, MSB):	verify error on data byte.
Bit 6 (**`0x40`**):	verify error on protection bit.
Bit 3 (**`0x08`**):	data byte already in cache.
Bit 2 (**`0x04`**):	write protection bit already in cache.
Bit 0 (**`0x01`**, LSB):	write protection bit set for this data byte.

Example: Read status information for 21 bytes starting at offset 16. The data any write protection bit are already present in the cache for every byte. Furthermore the first five bytes are write protected.

Command:	Str("**`Card,MemReadStatus,16,21`**")
DataIn:	nil
DataOut:	**`0x0D 0x0D 0x0D 0x0D 0x0D 0x0C 0x0C 0x0C 0x0C 0x0C 0x0C 0x0C 0x0C 0x0C 0x0C 0x0C 0x0C 0x0C 0x0C 0x0C 0x0C`**

Card,MemVerifyPin

Runs a PIN verification test of the card which may be required to get write or read access to the data contents. The PIN is given as a plain text string, and valid characters are "**0**" to "**9**" and "**A**A" to ”**A**".

Command:	Str("**Card,MemVerifyPin,<PIN>[,<Nr>]**")
DataIn:	nil
DataOut:	nil

<PIN>	The PIN.
<Nr>	Optional, number of PIN if card supports multiple PINs.

Important: If a wrong PIN is given too often, the card might be locked forever and thus become unusable. Please see the card’s manual for details.

Example1: Run a PIN verification with the PIN "**1234**".

Command:	Str("**Card,MemVerifyPin,1234**")
DataIn:	nil
DataOut:	nil

Example2: Run a PIN verification with "**89ABCD**" for PIN number 4.

Command:	Str("**Card,MemVerifyPin,98ABCD,4**")
DataIn:	nil
DataOut:	nil

Card,MemChangePin

Change the card’s PIN. The PIN is given as a plain text string and valid characters are "**0**" to "**9**" and "**A**" to "**F**".

Command:	Str("**Card,MemChangePin,<PIN>,<NewPIN>[,<Nr>]**")
DataIn:	nil
DataOut:	nil

<PIN>	The current PIN.
<NewPIN>	The new PIN.
<Nr>	**O**ptional, number of PIN if card supports multiple PINs.

Important: If a wrong PIN is given too often, the card might be locked forever and thus become unusable. Please see the card’s manual for details.

Example1: Change PIN from "**AB34**" to "**5678**".

Command:	Str("**Card,MemChangePin,AB34,5678**")
DataIn:	nil
DataOut:	nil

Example2: Change PIN number 2 from "**987F**" to "**CD12**"

Command:	Str("**Card,MemChangePin,987F,CD12,2**")
DataIn:	nil
DataOut:	nil

Card,MemSpecial

Returns a list of special commands which are supported by the current card. Currently, such commands are implemented for the chip types SLE4404, SLE4406 and SLE4436.

Command:	Str("**`Card,MemSpecial`**")
DataIn:	nil
DataOut:	Str("**`<Cmd1>[,<Cmd2>[...]]`**"
`<CmdX>`	Special command

Example: Special commands for a SLE4436

Command:	Str("**`Card,MemSpecial`**")
DataIn:	nil
DataOut:	Str("**`Deduct,ProgUser,ProgAuxData`**")

Warning: Sending the following commands to a card may lock it or even make it unusable. The commands will not be explained. Please refer to the data sheets for a detailed description. Towitoko can neither provide these data sheets nor offer any support about these commands.

The cache function is disabled for the following commands. **`DataIn`** must contain the entire card memory with changes. **`DataOut`** is empty (nil).
The first by of the card memory has the offset 0x00. Every byte is interpreted with the least significant bit first and the most significant bit last, i.e. 1100 1010 corresponds to 0x53.
Turning a bit from **`1`** to **`0`** is called writing and turning one from **`0`** to **`1`** is called erasing. In general, erasing is not always possible.

SLE4404: Verifying the correct User Code with **`Card,MemVerifyPin`** once is required before issuing one of the following SLE4404 specific commands. This also deletes the card's error counter.

Card,MemSpecial,ProgIssuerArea (SLE4404)

Allows modifying the Issuer Area (offset 0x02-0x07) while the Fuse is not blown.

Card,MemSpecial,ProgUserCode (SLE4404)

Allows changing the User Code (offset 0x08-0x09). The command **`Card,MemChangePin`** can also be used.

Card,MemSpecial,ProgErrorCounter (SLE4404)

Allows modifying the Error Counter area (offset 0x0A-0x0B). The real counter is located in the first four bytes at offset 0x0A. The remaining 12 bits are unused. The card becomes locked if the first four bits are each set to **`0`**.

Card,MemSpecial,ProgScratchPadMemory (SLE4404)

Allows modifying the Scratch Pad Memory area (offset 0x0C-0x0D).

Card,MemSpecial,ProgUserMemory (SLE4404)

Allows writing to the User Memory area (offset 0x0E-0x27). Bits can only turn from **`1`** to **`0`**. If the Fuse is blown, access may also depend on several other status bits.

Card,MemSpecial,ProgMemoryCode (SLE4404)

Allows modifying the Memory Code (offset 0x28-0x2B) while the Fuse is not blown. When the Fuse is blown, this command must be used to verify the Memory Code before erasing the User Memory area becomes possible.

Card,MemSpecial,ProgMemoryCounter (SLE4404)

Allows modifying the Memory Counter area (offset 0x2C-0x34). When the fuse is not blown, writing and erasing is possible. Otherwise bits can only be written, i.e. be turned from **`1`** to **`0`**.

Card,MemSpecial,EraseUserMemory (SLE4404)

After verifying the correct Memory Code with **`Card,MemSpecial,ProgMemoryCode`**, this command allows erasing the whole User Memory area. Every erasing attempt will also cause one bit in the Memory Counter area to be set from **`1`** to **`0`**. If no bit is left here, erasing becomes impossible.

Card,MemSpecial,ProgFuse (SLE4404)

Allows blowing the card's Fuse. Set bit 5 at offset 0x3E to **`0`** (value **`0xEF`**). After the fuse is blown, some memory is protected and can't be modified any longer. Blowing the Fuse is irreversible.

Card,MemSpecial,Deduct (SLE4406, SLE4436))

Allows writing to the Counter Area (offset 0x08- 0x0C). Bits can only turn from **`1`** to **`0`**, except on carry. This is detected and handled.

Card,MemSpecial,ProgUser (SLE4436)

Allows writing to the User Memory area (offset 0x28-0x2F), if possible. Bits can only turn from **`1`** to **`0`**.

Card,MemSpecial,ProgAuxData (SLE4436)

Allows writing to the Auxiliary Data area (offset 0x0E-0x0F). Bits can only turn from **`1`** to **`0`**.

Commad Set APPS

These commands include modules which provide an easy access to functions which are used frequently.

Apps,TLV

This module provides easy access to memory cards with a Tag-Length-Value (TLV) structure. This format uses the first byte for a tag ID or name. The second byte specifies the length of the data which starts at the third byte. If the bit 5 (**`0x40`**) is set in the tag name, the data itself contains another TLV structure – similar to a sub directory.

Apps,TLV,List

Returns a list of all TLV tags with complete path and data length. Name and length are separated by commas. List entries are separated by **`CR+LF`** (= **`#13#10`**). Length is given in decimal form.

Command:	Str("**`Apps,TLV,List`**")
DataIn:	nil
DataOut:	Str("**`<Tag1>,<Length1>#13#10[<Tag2>,<Length2>#13#10[...]]`**")
`<TagX>`	Path and name of tag.
`<LengthX>`	Length of data in bytes (decimal form).

Example: Tag **`0x61`** with length 10 contains two sub-tags: **`0x4F`** with 5 bytes of data and **`0x53`** with 1 bytes of data.

Command	Str(" **`Apps,TLV,List`**")
DataIn	nil
DataOut	Str(" **`61,10`** **`614f,5`** **`6153,1`**")

Apps,TLV,ReadTag

Read the data of a given tag.

Command:	Str(" **`Apps,TLV,ReadTag,<Tag>`** ")
DataIn:	nil
DataOut:	**`<TagData>`**
`<Tag>`	Path and name of a tag.
`<TagData>`	Data of this tag.

Example: Read tag **`0x61`**, it contains a sub directory with the tags **`0x4F`** with 5 data bytes and tag **`0x53`** with 1 data byte.

Command:	Str("**`Apps,TLV,ReadTag,61`**")
DataIn:	nil
DataOut:	**`0x4F 0x05 0x01 0x02 0x03 0x04 0x05 0x53 0x01 0x01`**

Apps,TLV,WriteTag

This command is currently not implemented.

Apps,TWK

Returns the decoded fields of a German prepaid telephone debit card.

Command:	Str("**`Apps,TWK[,<Field>]`**")
DataIn:	nil
DataOut:	Str("**`<Data1>#13#10[<Data2>#13#10[...]]`**")
`<Field>`	Data field, these names are valid: "**`Seriennummer`**", "**`Hersteller`**", "**`Datum`**", "**`Orginalwert`**". "**`Restwert`**", "**`Chipcode`**", "**`ChipHersteller`**", "**`Betreiber`**"
`<DataX>`	The data.

Note: The last two digits of the card's 9- or 11 digit serial number are only printed on the card, but not stored on the chip. Thus, there are 100 cards with an equal serial number.

Example 1: **`Apps,TWK`** returns all values, separated by the characters **`CR+LF`** (**`#13#10`**).

Command: Str("**`Apps,TWK`**")
DataIn nil
DataOut Str("

```
Seriennummer=131212752
Hersteller=Giesecke & Devrient, München
Datum=DEZ 19x3
Orginalwert=50,00 DM
Restwert=0,00 DM
Chipcode=1304
ChipHersteller=THOMSON
Betreiber=Deutsche Telekom AG
```

")

Example 2: If the command string is supplemented by a keyword, only the specified parameter is returned, e.g. only the remaining value.

Command Str("**`Apps,TWK,Restwert`**")
DataIn nil
DataOut Str("**`0,00 DM`**")

Apps,KVK

Decodes the fields of a German health insurance card. The data on this card is stored in a TLV structure.

Command: Str("**`Apps,KVK[,<Field>]`**")
DataIn: nil
DataOut: Str("**`<Data1>#13#10[<Data2>#13#10[...]]`**")

`<Field>` Data field, the following names are valid: "**`Krankenkasse`**"**`,`** "**`KNummer`**", "**`VkNr`**", "**`VNummer`**","**`Status, `**"**`StatusExt`**", "**`Titel`**"**`, `**"**`Vorname`**"**`,`** "**`Zusatz`**", "**`Name`**", "**`GebDatum`**", "**`Strasse`**", "**`Land`**", "**`PLZ`**", "**`Ort`**", "**`Gultigkeit`**"

`<DataX>` The data.

Example 1: **`Apps,KVK`** returns all values, separated by the characters **`CR+LF`** (**`#13#10`**).

Command: Str("**`Apps,TWK`**")
DataIn nil
DataOut Str("

```
Krankenkasse=Bundesknappschaft
KNummer=9905003
VkNr=74701
VNummer=1234567801
Status=1000
StatusExt=1
Titel=Dr.
Vorname=Martin
Zusatz=Baron
Name=Mustermann
GebDatum=12031960
Strasse=Alte Holstenstraße 46
Land=D
PLZ=21031
Ort=Hamburg
Gultigkeit=1201
```

")

Example 2: If the command string is supplemented by a keyword, only the specified parameter is returned, e.g. only the date until the card is valid.

Command Str("**`Apps,TWK,Gultigkeit`**")
DataIn nil
DataOut Str("**`1201`**")

Apps,ISO / Apps,ECB / Apps,GSM / Apps,TRP / Apps,PAY

These commands will not be implemented.

Command Tree

Here is a complete list of all SCardServer commands organized as a tree:

- System
  - Info
    - ErrCode
    - ErrText
    - Handle
    - Lng
    - UsedMemHeap
    - UsedMemTotal
    - VersionCode
    - VersionText
  - TaskList
  - Create
  - Destroy
  - TaskTitele
  - TaskPath
  - AddHWndMsg
  - DelHWnd
  - SetMainHWnd
  - SetLng
  - ConvertErrCode
  - Comands
  - CryptKey
    - DES
  - GenCryptKey
    - DES
- Device
  - Info
    - Status
    - Port
    - Type
    - ShortName
    - Index
    - Version
    - Serial
    - LotNr
    - Baudrate
    - MaxBaudrate
    - Led"
    - Caps
    - Mode
    - MouseDetect
    - PowerFail
  - CheckPowerFail
  - List
  - Refresh
  - Select
  - SearchComPort
  - Remove
  - InfoDeviceID
  - InfoDeviceIDCard
  - SetLed
  - SetMode
- Card
  - Info
    - Status
    - LockedBy
    - LinkerApps
    - LinkerCards
    - PtsAuto
    - PtsBinary
    - PtsBinaryLen
    - Baudrate
    - CardCount
    - CardPower
    - Type
    - Protocol
    - ATR
    - Apps
    - MemSize
    - PinSize
    - PinCnt
    - PageSize
    - ErrMem
    - ErrMemPB
    - AtrBinary
    - AtrBinarySize
    - AtrHistory
    - AtrHistorySize
    - TS, T0, TA1..TD8
    - SAD
    - DAD
    - IFSC
    - IFSD
    - CWT
    - BWT
  - Lock
  - Unlock
  - APDU
  - ISOAPDU
  - Reset
  - T0TX
  - T0RX
  - T1
  - PTS
  - TspTxRxLen
  - InitBwtCwt
  - InitSadDad
  - InitIfsdIfsc
  - MemDisableCache
  - MemEnableCache
  - MemRead
  - MemWrite
  - MemVerify
  - MemReadPB
  - MemWritePB
  - MemVerifyPB
  - MemSetPB
  - MemReadStatus
  - MemVerifyPin
  - MemChangePin
  - MemSpecial
    - Deduct
    - ProgUser
    - ProgAuxData
- Apps
  - TLV
    - List
    - ReadTag
    - WriteTag
  - TWK
    - Seriennummer
    - Hersteller
    - Datum
    - Orginalwert
    - Restwert
    - Chipcode
    - ChipHersteller
    - Betreiber
  - KVK
    - Krankenkasse
    - KNummer
    - VkNr
    - VNummer
    - Status
    - StatusExt
    - Titel
    - Vorname
    - Zusatz
    - Name
    - GebDatum
    - Strasse
    - Land
    - PLZ
    - Ort
    - Gultigkeit

Further Information Sources

Internet Pages

Semiconductor companies:

Atmel	http://www.atmel.com
Giesecke & Devrient	http://www.gdm.de
Hitachi	http://semiconductor.hitachi.com
Infineon (Siemens)	http://www.infineon.com
Motorola	http://www.mot-sps.com
Philips	http://www.semiconductors.philips.com
Samsung:	http://www.samsungsemi.com
ST Microelectronics (SGS Thomson)	http://www.st.com
Texas Instruments	http://www.ti.com
XICOR	http://www.xicor.com

More links leading to manufacturers, interfaces etc. can be found at the Towitoko homepage http://www.towitoko.de or at http://www.scdk.com

Literature

Wolfgang Rankl, Wolfgang Effing: *Handbuch der Chipkarten*
3. Auflage, März 1999, Carl Hanser, München, ISBN 3-446-21115-2

Francesco P. Volpe, Safinaz Volpe: *Chipkarten. Grundlagen, Technik, Anwendungen*
1996, Heinz Heise Verlag, Hannover, ISBN: 388229065X

Stefan Schütt, Bert Kohlgraf: *Chipkarten*
April 1996, R. Oldenbourg, München, ISBN 3-486-23738-1

Yahya Haghiri, Thomas Tarantino: *Vom Plastik zur Chipkarte*
November 1999, Carl Hanser, München, ISBN 3-446-21249-3

Patrick Horster: *Chipkarten*
1998, Vieweg, Wiesbaden, ISBN: 3528056673

Scott Guthery: *Smart Card Developer´s Kit*
Dezember 1997, Macmillan, ISBN 1-57870-027-2

Mike Hendry: *Smart Card Security and Applications*
September1997, Artech House Publishers; ISBN 0-89006-953-0

Dreifus Henry: *Smart Cards: A Guide to Building and Managing Smart Card Applications*
Dezember 1997, John Wiley & Sons, ISBN: 0471157481

KartenZwerg® und CHIPDRIVE™ are trademarks of Towitoko AG.
Delphi® is a trademark of Inprise Corporation.
Windows 3.1®, Windows 95®, Windows 98®, Windows ME®, Windows NT® and Windows 2000 are trademarks of Microsoft Corporation