

CardServer V2.02

Technical Documentation

SmartCard Manager, SCARD Interface, Delphi Component

Release 02.03.98

© '98 TOWITOKO electronics GmbH

Version history

V2.01

- Delphi component extended by following properties:
ConfigMenuItem, ConfigPopupMenu, ConfigMaxPort

V2.02

- Command 'Device, Select' was extended, so that alternately to the index of the device the device name (as retrieved with 'Device, Info') can be used.
- The CardServer task SCRDSERV.EXE / SCARDS32.EXE has to be located in the Windows directory. The program start invoked by all interfaces (e.g. SCARD.DLL) is with explicit path.
- Version check of the interfaces against the server task
- Messages for device detection and list management (tasks, devices) are added
- Delphi component extended by:
Events: OnDeviceListChange, OnTaskListChange, OnCardInfoChange, OnDeviceSearch
Properties: Enabled
- Support of new XICOR memory chips
- Full functionality with Windows NTTM 4.0
- Problem with accessing 'KartenZwerg' and 'CHIPDRIVE extern II' fixed
- Minor bugs fixed

CardServer - Overview

A large number of manufacturers of smartcards, terminals and drivers do exist. Also many industrial standards, card protocols and standardization have been set up. Our goal is to make the integration of smartcards and terminals into your application software as easy as possible. The CardServer will handle the following jobs for you:

Management of connected terminals:

- Management of a selection list for all connected smartcard terminals, similar to the selection lists for printers (e.g. 'CHIPDRIVE extern on COM 1')
- Status information on each terminal: Status of the smartcard, serial number, terminal information.
- The most recent configuration data is stored in a INI-file (e.g. COM port assignments)

Management of connected application programs:

- Management of a list with all application programs currently bound to the CardServer
- The CardServer gives control on the card exactly to one application program at a time. If the application software is finished with card access CardServer passes control to the next application program.
- CardServer can start specific application programs in dependence of the inserted card type if the application software has been registered for this type of card. Either the card application type (e.g. GSM or EC-card) or the AID of the card can be used as identifying mark. (from V2.10).

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 write and 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!

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:

- GSM cards according to GSM11.11 are recognized and can be accessed with macro functions, i.e. by issuing the command 'Apps,Gsm,ReadAdn,17' you can directly read the 17th short dial number entry. Equally of course you can process any T0 command. The GSM module supports the following data areas: ADN, FDN, MSISDN, LDN, BDN, SMS, SMSP, CBMI, PUCT, PLMN and many others
- German health insurance card 'KVK'
- German telephone prepaid debit card
- German EC-card with chip (national electronic purse)
- with PlugINs you can create your own card application modules! (from V.2.10)

Interfaces of the CardServer

The CardServer runs as a separate background task under Windows® 95, Windows NT™ and Windows® 3.11. Application programs can communicate with the CardServer using one of the following interfaces. All interfaces are compatible with all operating systems listed above.

TDEV interface - TDEV.DLL, TDEV32.DLL

The **TDEV** interface exists for compatibility our earlier driver support interface. We do recommend the use of the new SCARD interface because only that way you have full access to all new features of the Card Server.

Available in 16- and 32 Bit Version running under Windows® 95, Windows NT™ and Windows® 3.11.

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

CT-API interface, compatible with CT-API V1.1 (Issued by: Deutsche Telekom AG / PZ Telesec, GMD Forschungszentrum Informationstechnik GmbH, TÜV Informationstechnik GmbH und TeleTrustT Deutschland e.V.)

More details on this specification can be found on the word wide web at:

<http://www.darmstadt.gmd.de/~eckstein/CT/mkt.html#SPEK>.

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 CardServers functionality.

Available in 16- and 32 bit Version running under Windows® 95, Windows NT™ and Windows® 3.11.

SCARD interface - SCARD.DLL, SCARD32.DLL

The **SCARD** interface packages the full CardServer functionality. The implementation on client side is extremely easy. Only one single DLL-function call is used for all accesses. Windows messages do the event handling for your application program

Available in 16- and 32 bit Version running under Windows® 95, Windows NT™ and Windows® 3.11.

Delphi 1/2/3 component - SCARDCMP.PAS

For **DELPHI** we have a component available, which makes the implementation even simpler. All events are implemented and various lists (terminals, applications, terminal status information, card status information) are available in form of string lists.

Available in 16- and 32 bit Version running under Windows® 95, Windows NT™ and Windows® 3.11.

The SCARD interface

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

Both DLLs 16/32 BIT (SCARD.DLL / SCARD32.DLL) export the following command:

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

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

Handle	In case more instances of DLL are required by the application software 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 CardServer in this case will do the assignment via the thread- / task handle of your application program.
Cmd	CardServer command (zero terminated string)
CmdLen	this value contains the length of the command string, if the data transfer to the CardServer is encrypted. If un-encrypted transfer is used, this value must be set to zero,
DataIn	pointer on input data
DataInLen	length of the input data
DataOut	pointer on 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 if command execution was successful

Sample code for implementation with VB4/5, Access

```
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
```

Note: If you are using a 16-bit version of VB you must use the 16 bit version of the DLL: ... **Lib** "SCARD.DLL" ...

Sample code for implementation with PASCAL / DELPHI

```
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";
```

Note: Using Delphi 1 (16-bit version) you must use the 16 bit version of the DLL:
... LongInt); LongInt; **external** "SCARD";

The DELPHI component

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

- loads the SCARD Library (16/32) dynamically and imports the SCardComand function
- creates a object instance to the Card Server
- creates a window handle and registers it for the receipt of CardServer events
- introduces a new exception 'ESmartCard' and this way forwards error messages

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

TSmartCard - methods

```
function Command (const Cmd: string;
                  DataIn: pointer; DataInLen: longint;
                  DataOut: pointer; DataOutMax longint): longint
```

This method packages the SCardCommand function for communication with the CardServer. 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 bytes written to DataOut. If a error occurs a 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 form of a string list is placed in lines (e.g. used by DeviceList).

TSmartCard - PROPERTIES

Active: Boolean;

This property set to TRUE will cause the component to load the SCARD library and hereby starting the CardServer. FALSE will unload the library.

Enabled: Boolean;

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

DeviceInfo: TStringList

List of the terminal status information on the currently selected terminal

DeviceList: TStringList

List of all available terminals

TaskList: TStringList

List of all application programs / tasks bound to the CardServer

CardInfo: TStringList

List of status information on the currently inserted smartcard

AutoUnlock: Boolean

This property allows the automatically release of the terminal (command 'device, unlock') after ending the OnActiveCard event.

ConfigPopupMenu: TPopupMenu

This property assigned to an pupup menu inserts all necessary menu entries for the configuration of the CardServer and terminal selection. ConfigMenuItem is deleted if this property is used.

ConfigMenuItem: TMenuItem

If this property is assigned to an menu entry, the component automatically adds all necessary entries for the configuration of the CardServer and terminal selection. ConfigPopupMenu is deleted if this property is used.

ConfigMaxPort: Integer

This property denominates the maximum number of available COM-Ports in the ConfigMenu.

TSmartCard - EVENTS**OnDeviceError: TCardEvent**

Terminal access failed / the terminal to PC connection was interrupted!

OnCardWait: TCardEvent

No card is present in the terminal / the card has been removed from the terminal

OnCardDetect: TCardEvent

A card has been inserted to the terminal. The card cannot be accessed yet!

OnCardInvalid: TCardEvent

The card recognition has failed / no valid card!

OnCardActive: TCardEvent

The card was recognized and activated. The card can now be accessed.

OnCardLock: TCardLockEvent

Another application program has started to access the card

OnCardValid: TCardEvent

All applications are finished with the access to the card (command 'Card,Unlock'). It is now possible to access the card again

OnProgress: TProgressEvent

Event for reporting progress on memory card access

OnDeviceSearch: TSearchEvent

Event for display of progress during search for a terminal (started with command 'Device, SearchComPort' or at first start of the CardServer)

OnCardInfoChange: TNotifyEvent

Event for displaying new data in the CardInfo list

OnTaskListChange: TNotifyEvent

Event for displaying new data in the TaskList

OnDeviceListChange: TNotifyEvent

Event for displaying new Data in the DeviceList

CardServer Reference

For making the implementation of smartcard access as simple as possible the CardServer uses the same syntax for every command. The selection of the function calls and the transmission of eventually necessary parameters is done with a command string. Input and output data are optional.

A command string always contains key words and parameters separated by comma.

```
Example 1  command  Str( " Device,Info,Type " )
           DataIn   nil
           DataOut   Str( " CHIPDRIVE extern " )
```

The command returns the current terminal type.

Possible return codes: 0 = OK

```
Example 2  command  Str( " Card,MemWrite,16,8 " )
           DataIn   Str( " TOWITOKO " )
           DataOut   nil
```

This command writes 8 characters starting at address 16 to a memory card.

Possible return codes: 0='OK', 4000='No card present in terminal', 1009='Terminal is locked'

Get started quickly

For the development of the CardServer the easy implementation was one of the most important factors. The CardServer offers all functionality of the PC/SC standard (and even more) but still it is kept simple on the application programmers side so that you can get started with least effort right away.

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

But equally as important you gain access to a great number of powerful features which will especially be delighting for all professional users.

The card status

The CardServer handles the management of the card. For each application program this information on the status of the card and the terminal is held available:

- It is tested if the terminal is connected and responding properly. In case of an failure the status is set to **ERROR**
- It is tested if there is a card present in the terminal. In case no card is present 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 yet in this state (error 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 on the cards is given to exactly one application program. This application program receives the status **ACTIVE** while all other application programs 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 program.
- In case b) the CardServer passes control on the card to the next Application program, which again can pass on control to the next application.
- If all application programs 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 program.

- If a application program 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 program which issued the request - all other applications get the status **LOCKED**.
- The active application program may release the card by issuing a 'Card,Unlock' and the status for all application programs will return to **VALID**.

The Status can be polled with the commands 'Card,Info,Status' or 'Card,Info'.

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 un-register) any number of windows for the receipt of CardServer messages using the commands "System,AddHwndMsg" and "System,DelHwnd".

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

- In case of a status change (e.g. **WAIT** -> **DETECT**)
- Within the status **LOCKED** if control is passed to another application program

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 program 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.

Structure of the messages

The Windows message is sent to the given window handle using 'PostMessage'. The message ID (MsgID) can be specified by you with the registration of the window handle (compare System,AddHwndMsg,[Hwnd],[MsgID]).

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 program |
| • 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

Order of activation of application programs

The CardServer determines the order in which the application programs are assigned access to the card. The priority is determined by the following criteria (in order of the List) (from V2.10).

It is determined if:

- a 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).
- a application program 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.

In case several application programs have the same ranking or no criteria were matched the tab sequence of the Windows-desktop is used for determining the first application program.

Rules for smooth cooperation of multiple application programs

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

- register reliable criteria
- allow the CardServer to start up your application program 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 but better just add a line of text to a status line, e.g. 'Card is 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 ("Device,List").

By doing so the user will have all choices for:

- a) register new terminals - command "Device,SearchComPort,[Port]"
- b) use the 'automatic terminal selection' - command "Device,Select,-1"
- c) select a explicit terminal - command "Device,Select,[Index]"

Global Return Codes

An important advantage of the CardServer 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.

Code	Error message text	Description
0000	OK	Command was successfully executed
1001	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.
1002	serial port is used by another application	The COM-port is used by another application program (e.g. a mouse or a modem)
1008	no terminal detected on selected port	The COM-port configured properly but no terminal was detected. Check the connection!
1009	terminal is locked by -	At the moment access to the terminal is not possible because another application is accessing a card or has not released the card.
1010	The application software is not compatible with this terminal	TOWITOKO distributes OEM-terminals, which are not compatible with the TOWITOKO standard software products. The terminals can be upgraded for archiving compatibility. (see offers at http://www.towitoko.de)
4000	No card present in terminal	
4001	Card was removed during access	
4002	Invalid card present in terminal	
4004	Card ejection failed	Reserved for future terminals with automatic card ejection.
1200	unknown command	The command string was not recognized
1201	command execution not possible with current card	Not all commands can be used with all cards - especially between memory and processor cards.
1202	command execution not possible with this terminal	Occurs e.g. if a T0 command is sent to a terminal not supporting processor cards.
1203	invalid command parameter	e.g. invalid address range for the command 'Card,MemRead'
1310	smartcard access failed	A non recoverable error occurred during access to the smartcard
1311	PIN error! #1 trial(s) left	PIN-Error for memory smartcards; #1 is replaced by the remaining number of trials
2000	server not available	The CardServer failed to start

CardServer - command set SYSTEM

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

System,Info

Determines information on the CardServer and the status of the command execution. The following values can be retrieved.

VersionCode: Version of the CardServer (4 digit BCD encoding)

VersionText: Version as string.

ErrText: Text of the last error message.

ErrCode: Error code (global return code) of the last command causing an error.

Lng: preset language for the calling application program.

Handle: Handle, assigned to the calling object instance.

The function call "**System,Info**" returns all values (separated by CR/LF = #13#10). Is the command string supplemented by a keyword, only the specified parameter is returned.

```
Example 1:  Command  Str( " System,Info " )
            DataIn   nil
            DataOut   Str( " Handle=3
                           Lng=GERMAN
                           VersionCode=0202
                           VersionText=TDEV-Server V2.02
                           ErrCode=4002
                           ErrText=Ungültige Karte im Lesegerät " )
```

```
Example 2:  Command  Str( " System,Info,Lng " )
            DataIn   nil
            DataOut   Str( " GERMAN " )
```

System,TaskList

Returns the list of application programmes currently connected to the CardServer:

```
Example:    Command  Str( " System,TaskList " )
            DataIn   nil
            DataOut   Str( " Delphi,'CHIPDRIVE micro' an COM2
                           Testgo,'CHIPDRIVE micro' an COM2 " )
```

System,Create

The CardServer creates an instance for every connected application program. Herefore the taskhandle of the application is used. It is not necessary to create a object instance, if only one instance is needed.

If several instances of a application are needed, they have to be set up with the 'System,Create' command.

Important: The parameter handle needs to be set to -1 for calling

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

System,Destroy

Releases a object instance which was generated with the Create command. The CardServer automatically activates this function, if the task handle of the application gets invalid, i.e. the application was closed.

Example. Command Str(" **System,Destroy**,[handle] ")
 DataIn nil
 DataOut nil

handle Handle, which was generated with "**System.Create**" earlier

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.

Example Command Str(" **System,AddHwndMsg**,[hwnd],[msgID] ")
 DataIn nil
 DataOut nil

hwnd Window handle of window to receive messages

msgID Message value for the notification (Message ID)

System,DelHwnd

Deletes a window handle from the list.

Example. Command Str(" **System,DelHwnd**,[hwnd] ")
 DataIn nil
 DataOut nil

hwnd Windows handle of the windows that had recieved the messages

System,SetLng

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

Example Command Str(" **System,SetLng**,[lngstr] ")
 DataIn nil
 DataOut nil

lngstr Language (= Section string in the file SCARD.ERR) e.g. 'ENGLISH'

System,ConvertErrCode

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

Example. Command Str(" **System,ConvertErrCode**,4002 ")
 DataIn nil
 DataOut Str(" **Invalid Card in terminal** ")

System,Comands

Returns a List of all available commands. The command tree can be listed recursively by adding more keywords:

Example 1: Command Str(" **system,Comands** ")
 DataIn nil
 DataOut Str(" **system**
 Device
 Card
 Apps")

Example 2: Command Str(" **system,Comands,Apps** ")
 DataIn nil
 DataOut Str(" **List**
 Info
 ...
 SetLedCard")

System,CryptKey

This command activates the encrypted communication with the CardServer. Command string and DataIn need to be presented in encrypted form after issuing this command. Correspondingly DataOut is returned in encrypted format by the CardServer. The algorithm is a standard DES.

Important: Since the length of data always is 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
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.

Example. Command Str(" **system,CryptKey,DES** ")
 DataIn **KeyID** (8 Byte)
 DataOut nil

KeyID The DES-key is not transmitted directly but in encrypted form. More details on this are found in section GenCryptKey

System,GenCryptKey

Of course the encryption only makes sense, if the key itself is not transmitted itself. 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.

Example Command Str(" **system,GenCryptKey,DES** ")
 DataIn **DES-Key** (8 Byte)
 DataOut **KeyID** (8 Byte)

DES-Key DES-key used for actual encryption of the data

KeyID This value is needed in the application phase.

System,Upgrade

An upgrade of the terminal hardware (OEM terminals) allows full compatibility with TOWITOKOs standard software packages. More detailed information on upgrading is available at <http://www.towitoko.de>. The upgrade code is saved to the INI file of the Card Server. In case a new installation of the software is necessary, the upgrade code has to be entered again.

Example.	Command	Str(" System,Upgrade,[Lizenz] ")
	DataIn	nil
	DataOut	nil
Lizenz		License number

System,OemRegister

If you are developing application software for a OEM-Terminal you receive a unique driver certificate which you have to register with this command prior to accessing the terminal.

Example	Command	Str(" System,OemRegister,OemID ")
	DataIn	nil
	DataOut	nil
OemID		Driver certificate

CardServer - Command Set DEVICE

Device,Info

Returns a list of all terminal parameters. The information relate to the terminal currently assigned to the application (see 'Device,Select'):

status: indicates the terminal status:
 error terminal inaccessible
 valid terminal ready

Port: COM-port on which the terminal is connected

Type: device name, e.g. 'CHIPDRIVE twin Slot 1'

ShortName: short name of the terminal type
 CDX CHIPDRIVE extern
 CDM CHIPDRIVE micro
 CDI CHIPDRIVE intern
 CDD CHIPDRIVE extern II
 CD1 / CD2 CHIPDRIVE twin slot 1 / 2
 KTZ KartenZwerg (OEM Version)
 CCR CardReader (OEM Version)

Serial: serial number of the terminal within the production lot number

LotNr: production lot number of the terminal

Version: hardware revision number

Baudrate: COM-port transmission speed

Led: status display (see 'Device,SetLed')

The function call "**Device,Info**" returns all values (separated by CR/LF = #13#10). If the command string is supplemented by a keyword only the specified parameter is returned.

```
Example 1  command  Str( " Device,Info" )
           DataIn   nil
           DataOut   Str( " status=valid
                           Port=COM2
                           Type=CHIPDRIVE micro
                           ShortName=CDM
                           Version=1.1
                           LotNr=9805
                           Serial=861
                           Baudrate=9600 " )
```

```
Example 2  command  Str( " Device,Info,LotNr " )
           DataIn   nil
           DataOut   Str( " 9805 " )
```

Device,InfoDeviceID

Command similar to 'Device,Info' but relating to a specified terminal within the terminal list ('Device,List')

```
Example    Command  Str( "Device,InfoDeviceID,[DevID],Serial " )
           DataIn   nil
           DataOut   Str( " 861" )
```

DevID terminal index (zero = first entry in the terminal list)

Device,List

returns a list of all terminals connected to the CardServer:

Example	Command	Str(" Device,List")
	DataIn	nil
	DataOut	Str(" 'CHIPDRIVE micro' an COM2 'CHIPDRIVE twin Slot 2' an COM3 'CHIPDRIVE twin Slot 1' an COM3 ")

Device,Select

Using this command you select a specific terminal from the terminal list or you activate the automatic terminal selection.

Example.	Command	Str("Device,Select,[DevID] ")
	DataIn	nil
	DataOut	nil

DevID terminal index (zero = first entry in the terminal list) or name

If DevID is set to -1 or the command 'Device,Select' has not been issued the automatic terminal selection is activated. The following criteria apply:

- 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 program and remains assigned until the card is removed.

Alternately terminals can be selected with this command by name, short name or COM-Port. This is the preferred choice if the selection of terminals is to be stored in a INI-file of the application program since the order of the devices within the list may change.

Valid samples for DevID are:

"CHIPDRIVE twin Slot 1 an COM1"	Selection by name and COM-Port (unique)
"COM1"	Selection by COM-Port
"CHIPDRIVE extern"	Selection by name
"CDX"	Selection by short name
"CD1 COM3"	Selection by short name and COM-Port (unique)
"2"	Selection by Index (unique)

If the selection is not unique, the first device in the list with matching description is used.

Device,Remove

Use this command for removing a terminal from the terminal list, i.e. the associated COM port is released. The terminal can be reconnected with 'Device,SearchComPort'

Example	Command	Str("Device,Remove,[DevID] ")
	DataIn	nil
	DataOut	nil

DevID terminal index (zero = first entry of the terminal list)

Device,SearchComPort

Use this command for initiating a search on a terminal device on the indicated COM-Port. If a terminal is detected, the CardServer determines all device specific data such as device type and serial number. Functional devices are stored in the INI file of the CardServer. On the next start of the CardServer previously detected devices are again tested and installed.

Example	Command	Str("Device,SearchComPort,[Port] ")
	DataIn	nil
	DataOut	nil

Port number of the COM port on which the terminal is connected. If no parameter is assigned all COM ports not used otherwise are searched.

Device,SetLed

This command controls the status display of the terminals. The command relates to the active terminal of your application program.

Example	Command	Str("Device,SetLed,[ColorStr] ")
	DataIn	nil
	DataOut	nil

ColorStr the string contains up to 8 characters, each assigned to a color:
0=off, 1=red, 2=green, 3=yellow.
Examples:
"01" fast red blink signal
"0011" slow red blink signal
"2" steady green signal
"123" red, green, yellow cycling

CardServer - Command Set CARD

Card,Info

Returns a list of status information on the currently inserted card:

Status	returns the status of the card:
error	terminal inaccessible
wait	no card in reader
detect	card was inserted and is analyzed
invalid	card is invalid / card type was not detected
valid	card is valid and currently not processed by any application
active	card is valid and currently processed by your application
locked	card is valid and currently processed by another application
LockedBy:	Index of the active application within the application list (System,Tasklist) followed by the application name (as string, separated by a comma)
Type	The exact type declaration of supported memory cards. Following is a list of the currently implemented chip types. The list is constantly updated. The latest information is available on the world wide web at http://www.towitoko.de SLE4404, SLE4406, SLE4436 SLE4432, SLE4442, SLE4428, SLE4418, SLE4418K SC152, GPM896 I2C 2K, I2C 4K, I2C 8K, I2C 16K I2C 32K, I2C 64K, I2C 128K, I2C 256K, I2C 512K X24165, X24645, X76F041, X76F100, X76F128, X76F640 MCM2814ATR CPU
Protocol	indicates the protocol of the card: ATR cards with special bit protocols e.g. 4406/4436 2W 2-wire protocol 3W 3-wire protocol I2C I2C-bus protocol I2CX I2C-bus protocol with 2 byte addressing XC... special I2C-bus protocol for XICOR chips T0, T1, T14 processor card protocols
Apps	return a list of detected card application modules (separated by comma) KVK valid German health insurance card TWK German prepaid telecom debit card GSM GSM carte according to GSM11.11 ECB German EC-card with chip (electronic purse) TRP TripleCard multi functional card PAY German PAY-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	memory cards I2C only: page size for write commands
ErrMem	memory cards only: number of errors occurred during write and verify commands
ErrMemPB	memory cards only: number of errors occurred during write and verify

commands
(write protection bits only)

AtrBinary	ATR in binary form
AtrBinarySize	size of the ATR in byte
AtrHistory	processor cards only: history bytes according to ISO7813-3
AtrHistorySize	size of the history in byte
TS, T0, TA1..8, TB1..8, TC1..8, TD1..8	decoded ATR according to ISO7816-3
SAD, DAD	processor cards T1 only: source- and destination addresses
IFSC, IFSD	processor cards T1 only: buffer size of the card and terminal
CWT, BWT	processor cards only: character- and block waiting time

The function call '**Card,Info**' returns all values (separated by CR/LF = #13#10). By supplementing the command string with an additional keyword only the specified parameter is returned:

```
Example 1   Command  Str( " Card,Info" )
            DataIn   nil
            DataOut   Str( " Status=active
                          LockedBy=0,Value Card Station
                          Type=CPU
                          Protocol=T0
                          CWT=1000
                          AtrBinarySize=8
                          AtrBinary=3B 85 00 54 53 2D 32 10
                          AtrHistorySize=5
                          AtrHistory=54 53 2D 32 10
                          TS=3B
                          T0=85
                          TD1=00" )
```

```
Example 2   command  Str( " Card,Info,Type" )
            DataIn   nil
            DataOut   Str( " CPU" )
```

```
Example 3   command  Str( " Card,Info " )
            DataIn   nil
            DataOut   Str( " Status=active
                          LockedBy=0,Value Card Station
                          Type=SLE4432
                          Protocol=2W
                          Apps=TLV,KVK
                          MemSize=256
                          AtrBinarySize=4
                          AtrBinary=A2 13 10 91" )
```

Card,Lock

Locks a card for access by other applications. The command can only be executed if a valid card is present in the terminal and no other application program currently is processing this card (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.

Example	Command	Str("Card,Lock")
	DataIn	nil
	DataOut	nil

Card,Unlock

This command is used to release a card for processing by other applications. The CardServer assigns the card to the next application program.

Example	Command	Str("Card,Unlock")
	DataIn	nil
	DataOut	nil

Card,APDU

This command sends a APDU to the card and receives the response of the card. The translation to the T0- / T1- protocol are done according to ISO7816-4. 'Case 1', 'Case 2 short' up to 'Case 4 short' with maximum data length of 254 bytes are supported.

Example 1, ISO CASE 1, command without data

Command	Str(" Card,APDU")
DataIn	CLA, INS, P1, P2
DataOut	SW1, SW2

Example 2, ISO CASE 2, command with data block from the card (0<= LE <= 255)

Command	Str(" Card,APDU")
DataIn	CLA, INS, P1, P2, Le
DataOut	SW1, SW2, data block

Example 3, ISO CASE 3, command with data block to the card (Lc > 0!)

Command	Str(" Card,APDU")
DataIn	CLA, INS, P1, P2, Lc, data block
DataOut	SW1, SW2

Example, ISO CASE 4, command with data blocks to and from card (LC > 0!)

Command	Str(" Card,APDU")
DataIn	CLA, INS, P1, P2, Lc, data block, Le
DataOut	SW1, SW2, data block

Lc Number of bytes to transmit to the card

Le Number of bytes expected from the card,
or zero for all data bytes (card determines length of response)

The return codes 9Fxx (GSM response data), 61xx, 6Cxx are not interpreted. This complies with the CT-API specification of a APDU and is not conforming with ISO7816-4.

Card,ISOAPDU

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

Note: GSM-cards operate with the T0-protocol but are (unfortunately) not compatible with ISO-standard in respect to the APDU since the return code 9Fxx is used instead of 61xx. The command section 'Apps,GSM,APDU' contains the correct APDU implementation for GSM-cards.

Card,Reset

This command initiates a hardware reset of the card. Eventually obtained access rights are lost.

Example.	Command	Str(" Card,Reset ")
	DataIn	nil
	DataOut	nil

Card,T0TX

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

Example	Command	Str(" Card,T0TX ")
	DataIn	CLA, INS, P1, P2, P3, data block
	DataOut	SW1, SW2

Card,T0RX

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

Example	Command	Str(" Card,T0RX ")
	DataIn	CLA, INS, P1, P2, P3
	DataOut	SW1, SW2, data block

Card,T1

This command executes a T1 command (including chaining if necessary):

Example	Command	Str(" Card,T1 ")
	DataIn	CLA, INS, P1, P2, data block
	DataOut	SW1, SW2, data block

The input data is sent transparently to the card. The nomenclature (CLA, INS ...) is according to ISO7816-3. Lc and Le are encoded according to "Case 1 to 4" specified in ISO7816-4 or defined in the specification of the card.

Card,TspTxRxLen

This command 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.

Example	Command	Str(" Card,TspRxLen,[RxLen] ")
	DataIn	data to be sent to card
	DataOut	data to be received from the card

RxLen This parameter specifies the number of expected bytes to be received from the card.

Card,InitBwtCwt

Block Waiting Time and Character Waiting Time can be overwritten, i.e. set manually. The initial values are taken from the ATR of the card.

Example. Command Str(" Card,InitBwtCwt,[Bwt],[Cwt] ")
 DataIn nil
 DataOut nil

Bwt Block Waiting Time, time-out of the first character of a block in [ms]

Cwt Character Waiting Time, time-out for the following characters in [ms]
 valid range: 1 to 60.000 (1 ms to 60 seconds.)

Card,InitSadDad

Initializes the SAD (Source Address) and DAD (Destination Address) for the T1 protocol.

Initially both values are set to zero. The syntax is same as for the command "Card,InitBwtCwt".
The valid range is 0 to 255.

Card,InitIfsdIfsc

Initializes IFSD (buffer size terminal) and IFSC (buffer size smartcard) for the T1 protocol. The initial values are taken from the ATR of the card. The syntax is same as for the command "Card,InitBwtCwt". The valid range is 0 to 255.

Card,MemDisableCache

This command disables the cache function for memory cards, i.e. even data already read is physically read again from the card for each access.

Example.	Command	Str(" Card,MemDisableCache ")
	DataIn	nil
	DataOut	nil

Card,MemEnableCache

Enables the cache function for memory cards. This is the default setting.

Card,MemRead

Reads the selected area from the data memory of a memory card, independent of the chip type:

Example	Command	Str(" Card,MemRead , [Adr], [Len] ")
	DataIn	nil
	DataOut	data read

Adr offset address on the smartcard for the first byte to read

Len number of bytes to read

Card,MemWrite

Writes the selected data area to the data memory of a memory card, independent of the chip type:

Example	Command	Str(" Card,MemWrite , [Adr], [Len] ")
	DataIn	data to write
	DataOut	nil

Adr offset address on the smartcard for the first byte to write

Len number of bytes to write

Note: Every write access is (internally) followed by an verify command. By using the command 'Card,MemReadStatus' you can retrieve the exact result of the write access (in case of an write error).

Note: If the cache function is active (default), only data bytes which have actually changed are written to the card. (only if the same data areas have been previously read from the card).

Card,MemVerify

This command performs an byte on byte comparison between the transmitted data and the data stored on the memory smartcard:

Example	Command	Str(" Card,MemVerify , [Adr], [Len] ")
	DataIn	data bytes, to be compared with the card data
	DataOut	nil

Adr offset address on the card

Len number of bytes to compare

Note: The number of errors on data bytes and write protect bits can also be retrieved with 'Card,Info'. In case of a verify error the error code 1310 „Card access failed“ is returned. Using the command 'Card,MemReadStatus' the exact result of the comparison can be retrieved.

Card,MemReadPB / Card,MemWritePB / Card,MemVerifyPB

These three commands are similar to the previous three commands but the functions relate not to the data memory but 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.

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:

00 hex write protection is not active

01 hex write protection is 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.

Example	Command	Str(" Card,MemSetPB ,[Adr],[Len] ")
	DataIn	nil
	DataOut	nil

The write protection for **Len** bytes starting at address **Adr** on the card is activated.

Card,MemReadStatus

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

Example	Command	Str(" Card,MemVerify ,[Adr],[Len] ")
	DataIn	status bytes
	DataOut	nil

The status information is encoded as follows:

Bit 7 (MSB)	1: verify error on data byte
Bit 6	1: verify error on write protection bit
Bit 3	1: data byte content is in cache memory
Bit 2	1: write protection bit content is in cache memory
Bit 0 (LSB)	1: write protect bit for corresponding data byte is active

Card,MemVerifyPin

Runs a PIN verification test of the card which may be required before write or read access to the data contents of the card:

Example	Command	Str(" Card,MemVerifyPin ,[PIN],[Nr] ")
	DataIn	nil
	DataOut	nil

PIN The BCD encoded PIN (e.g. 4 digits correspond 2 bytes)

Nr Selection of the PIN in case a card supports more than one PIN.
This parameter is optional.

Card,MemChangePin

Changes the PIN of a card:

Example.	Command	Str(" Card,MemChangePin ,[PIN],[NewPIN],[Nr] ")
	DataIn	nil
	DataOut	nil

PIN The BCD-encoded, current PIN

NewPIN The BCD-encoded, new PIN

Nr Selection of the PIN in case a card supports more than one PIN.
This parameter is optional.

CardServer - Command Set APPS

Apps,TLV,List

Returns a directory of all TLV tags with complete directory path (only possible for memory cards).

Example	Command	Str(" Apps,TLV,List ")
	DataIn	nil
	DataOut	Str(" 61,11 614f,6 6153,1 ")

The return data string contains a list of all tags on the card separated by CR/LF. Each line contains a tag (hex) and its length (decimal), separated by comma. The tag is preceded by its path.

Apps,TLV,ReadTag

Reads the contents of a tag (only possible for memory cards).

Example.	Command	Str(" Apps,TLV,ReadTag,[Tag] ")
	DataIn	nil or source
	DataOut	contents of the data field without tag and length

Tag	tag with path as described at command "Apps,TLV,List"
------------	---

Apps,TLV,WriteTag

Writes the contents of a tag and if necessary, creates the tag (only possible for memory cards). If the length of the new tag is different from the old one large data areas of the card may need to be rewritten.

Example	Command	Str(" Apps,TLV,WriteTag,[Tag] ")
	DataIn	new data content of data field without tag and length
	DataOut	nil

Tag	tag with path as described at command "Apps,TLV,List"
------------	---

Apps,ISO

In preparation. Commands according to ISO7816-4 will be directly executable.

Apps,ECB

In preparation. Data fields on a EC-card with chip will be accessed with these commands. Transaction commands are also in planning.

Apps,TRP

In preparation. Gives access to the TripleCard system for your application programmes.

Apps,GSM

In preparation. Immediate acces to: ADN,FDN,MSISDN,BDN,LDN,SMS,SMSP,CBMI,PLMN PUCT etc.

Apps,TWK,Read

Returns the data fields of a German telecom prepaid debit card in decoded format:

Serialnumber 9 digits of the 11 digit serial number of the card

Manufacturer card manufacturer

Date date of chip manufacturing

Original value original prepaid value of the card

Remaining value remaining (current) value of the card

The function call "**Apps,TWK,Read** " returns all values (separated by CR/LF = #13#10). By adding a keyword to the command string only the corresponding parameter value is returned.

```
Example 1  Command  Str( " Apps,TWK,Read " )
           DataIn   nil
           DataOut   Str( " Seriennummer=131212752xx
                        Hersteller=Giesecke & Devrient, München
                        Datum=DEZ 19x3
                        Originalwert=50,00 DM
                        Restwert= 0,00 DM" )
```

```
Example 2  Command  Str( " Apps,TWK,Read,Restwert " )
           DataIn   nil
           DataOut   Str( " 0,00 DM " )
```

Apps,KVK,Read

Returns the data fields of a German health insurance card in decoded and validated format:

Krankenkasse, KNummer, VKNr, VNummer, Status, StatusExt

Titel, Vorname, Zusatz, Name, GebDatum

Strasse, Land, PLZ, Ort, Gultigkeit

The function call "**Apps,KVK,Read** " returns all values (separated by CR/LF = #13#10). By adding a keyword to the command string only the corresponding parameter value is returned.

CardServer Command Tree

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

System	Info	VersionCode	Card	Reset	
		VersionText ErrText ErrCode Lng Handle		Lock Unlock T0TX T0RX T1	
System	Comands	ConvertErrCode		TspRxClen	
		TaskList		InitBwtCwt	
		Destroy		InitSadDad	
		AddHWndMsg		InitIfsdlfsc	
		DelHWnd		MemDisableCache	
		SetLng		MemEnableCache	
		CryptKey		MemRead	
		GenCryptKey		MemWrite	
		Upgrade		MemVerify	
		OemRegister		MemReadPB	
Device	Info	Status		MemWritePB	
		Port		MemSetPB	
		Type		MemVerifyPB	
		ShortName		MemReadStatus	
		Serial		MemVerifyPin	
		LotNr		MemChangePin	
		Version	Apps	TLV	List
		Baudrate			
		Led			
Device	InfoDeviceID		Apps	TWK	Read
	List				
	Select				
	Remove				
	SearchComPort				
	SetLed				
Card	Info	Status	Apps	KVK	Read
		LockedBy			
		Apps			
		Type			
		Protocol			Krankenkasse
		MemSize			
		PinSize			KNummer, VKNr
		PinCnt			
		PageSize			VNummer
		ErrMem			
		ErrMemPB			Status, StatusExt
		AtrBinary, AtrBinarySize			
		AtrHistory, AtrHistorySize			Titel, Vorname
		SAD, DAD, CWT, BWT			
		IFSC, IFSD			Zusatz, Name
		TS, T0, TA1-8, TB1-8, TC1-8, TD1-8			
					GebDatum
					Strasse, Land, PLZ, Ort
					Gultigkeit

ChipDrive™ is a trademark from TOWITOKO
KartenZwerg® is a registered trademark from TOWITOKO
Windows® is a registered trademark from Microsoft
Windows® 95 is a registered trademark from Microsoft
Windows NT™ is a trademark from Microsoft
Windows® 3.11 is a registered trademark from Microsoft