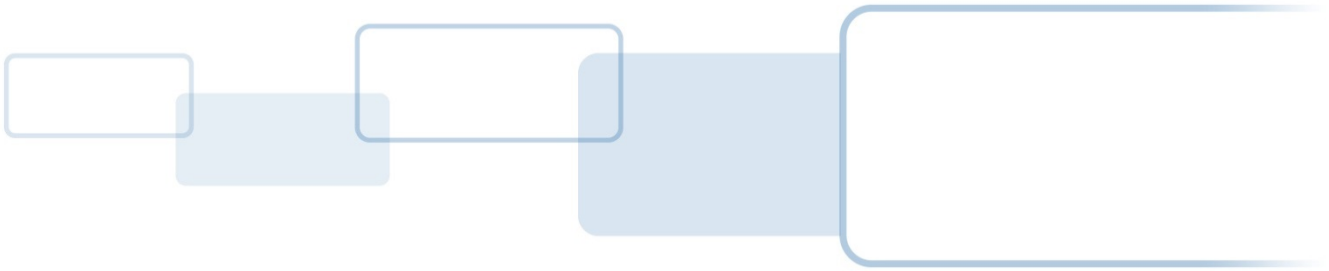
CHERRY KC 1000 SC
Software Developer Guide

PLT-02339, Rev. A.2
March 2015



hidglobal.com

Contents

Copyright

Trademarks

Revision History

Date	Author	Description	Version
03/2015	M. Chmielewski	Device configuration updated and restructured	A.2
11/2014	M. Goralczyk	Document update	A.1
07/2014	M. Chmielewski	Initial Version	A.0

Contacts

For additional offices around the world, see www.hidglobal.comcorporate offices.

North America & Corporate

611 Center Ridge Drive
Austin, TX 78753
USA
Phone: 866-607-7339
Fax: 949 732 2120

Asia Pacific

19/F 625 King's Road
North Point, Island East
Hong Kong
Phone: 852 3160 9833
Fax: 852 3160 4809

Europe, Middle East and Africa

Haverhill Business Park Phoenix Road
Haverhill, Suffolk CB9 7AE
England
Phone: 44 (0) 1440 711 822
Fax: 44 (0) 1440 714 840

About this Guide

Purpose

This Developer Guide is for developers integrating ISO/IEC 7816 contact cards using the CHERRY® KC 1000 SC.

Organization

1 Overview

1.1 Product Description

The CHERRY KC 1000 SC opens new market opportunities for system integrators seeking simple reader integration and development using standard interfaces, such as CCID (Circuit Card Interface Device). This reader works without installing or maintaining device drivers; only an operating system driver, for example, Microsoft CCID driver is necessary.

The CHERRY KC 1000 SC features include supporting the ISO/IEC 7816-3 card technology including synchronous cards support and Secure Pin Entry.

1.2 Features

- **CCID Support.** Removes the requirement to install drivers on standard operating systems to fully support capabilities of the reader board.
- **ISO/IEC 7816-3.** Supports the common contact card technology including synchronous cards support.
- **SPE.** Secure Pin Entry according to CCID and PC/SC specification.
- **Rapid and Easy Integration.** No special driver installation is required.
- **Advanced Power Management.** Fully compliant to Low Power modes specified by USB include the following.
  - Allows the computer to turn off the reader to save power (while the reader is still able to read cards, with reduced power)
  - Allows the reader to wake the computer

2 Getting Started

2.1 Driver Installation

As stated previously, no extra driver installation is necessary and every CCID compliant driver should work with the reader. However, Microsoft's CCID driver prevents the execution of CCID Escape commands. If an application uses those commands, apply the procedure in Appendix *A - Enabling Escape CCID commands*.

2.2 HID OMNIKEY Workbench

At present product is not supported by the HID OMNIKEY Workbench. Support shall be added in future releases of the tool.

3 Host Interfaces

3.1 USB

CHERRY KC 1000 SC supports USB 2.0 Full Speed (12 Mbits/s) interface.

The device enumerates as a composite device. USB protocol stack implements the following device classes:

- CCID (Integrated Circuit Cards Interface Device, v1.1)
- HID Keyboard

3.1.1 Endpoints Assignment

The table below lists the USB protocol stack endpoints, their parameters and functional assignment:

Endpoint	Description	Type	Max Packet Size
EP0	Control Endpoint	Interrupt IN/OUT	8
EP1	Keyboard	Interrupt IN	8
EP2	Smart Card	Bulk OUT	64
EP3	Smart Card	Bulk IN	64
EP4	Smart Card	Interrupt IN	8
EP5	Keyboard ACAPI (Media & Power Keys)	Interrupt IN	3

4 Human Interface

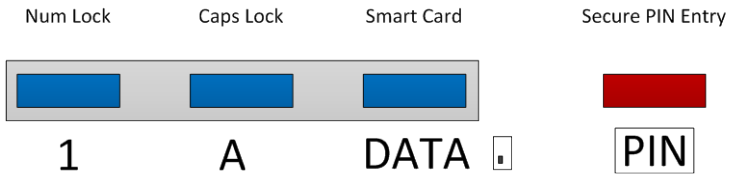
4.1 LEDs

There are 4 LEDs to indicate current status of the keyboard and smart card reader:

- Num Lock
- Caps Lock
- DATA
- PIN

Notice there is no Scroll Lock LED although all functions of Scroll Lock key are supported.

4.1.1 LED location



4.1.2 Num Lock

This LED indicates current function of numeric keypad. Its state is controlled by operating system and cannot be modified programmatically.

4.1.3 Caps Lock

Caps Lock indicates input mode in which all typed keys are uppercase. Its state is controlled by operating system and cannot be modified programmatically.

4.1.4 DATA

This is smart card status LED. It is ON when the smart card is powered. LED is OFF when smart card is not powered. It blinks when smart card reader transmits or receives any data to/from host computer.

Typically operating system powers smart card when it is inserted into slot. DATA LED is ON for that case. After few seconds, if there is no any application that makes use of smart card, LED is OFF indicating card power is off. If application communicates with smart card, LED stays ON blinking to indicate data transmission.

Function of this LED cannot be modified. Only blinking parameters like frequency or period may be altered, see User Hardware Configuration chapter for more details.

4.1.5 PIN

PIN LED indicates secure PIN entry mode by blinking. In this mode user is expected to enter PIN on numeric keypad. All other keys are ignored; keys pressed by the user on keypad are not reported to PC.

Secure PIN entry mode is started by one of secure pin commands. For more details see Secure Pin Entry Support chapter.

There is no possibility to control this LED programmatically other than starting secure pin entry session.

5 Keyboard

The CHERRY KC 1000 SC supports keyboard interface according to USB Device Class Definition for Human Interface Devices (HID Specification) version 1.11. It implements two USB protocol stack interfaces:

- Keyboard with BIOS support
- Media & Power Keys interface according to Advanced Configuration and Power Interface Specification (ACAPI)

6 Contact Card Interface

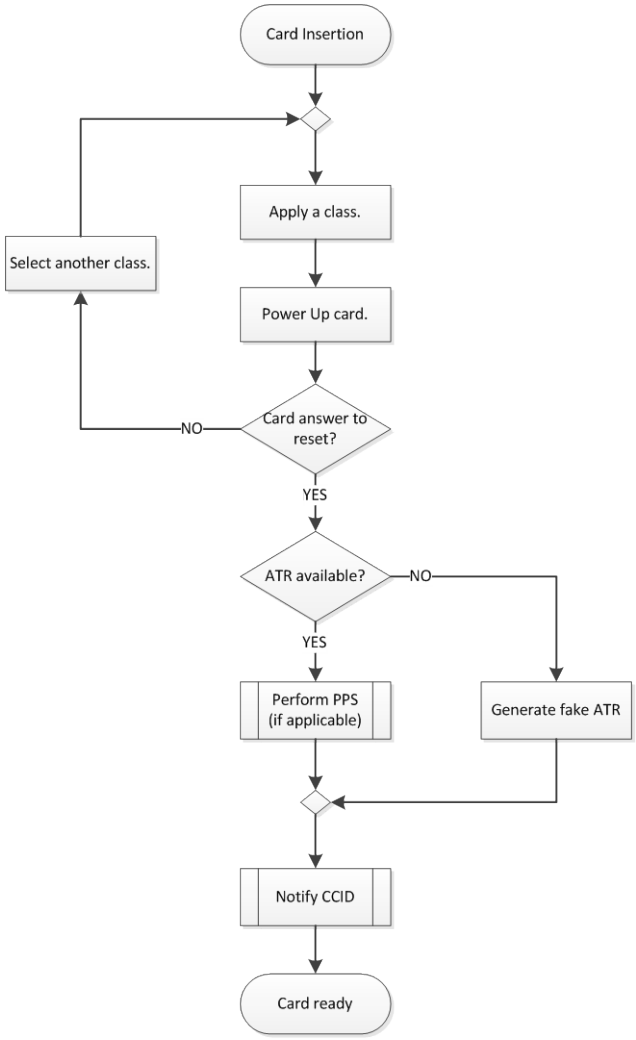
CHERRY KC 1000 SC is compliant with CCID specification [CCID]. It features following functionality:

- Configurable Voltage Selection
- Configurable Data Exchange Level: TPDU, APDU, extended APDU (for T = 1 cards only)
- Secure Pin Entry
- Configurable Operation Mode: ISO or EMVCo

6.1 Card Activation

Before card is ready for data exchange it must be properly activated. Diagram below shows the activation sequence:

- Card is inserted into a contact slot.
- Reader selects supply voltage, and powers up the card.
- If there is no response from the card another class is selected and the power up sequence is repeated. The sequence of voltage selection is configurable with **voltageSelection** parameter. If the card responds the response is evaluated.
- If the card returns valid ATR, PPS procedure follows (if applicable). If there is no ATR (i.e. some I^2C card), the state of lines is checked and the fake ATR composed. Finally the reader notifies CCID about the card presence.


Card Insertion
Apply a class.
Power Up card.
Card answer to reset?
NO
Select another class.
YES
ATR available?
NO
YES
Perform PPS (if applicable)
Generate fake ATR
Notify CCID
Card ready

6.2 Voltage Selection

CHERRY KC 1000 SC smartcard interface supports all classes listed in ISO/IEC 7816-3.

It is possible to set the sequence followed during voltage selection process. This can be used for a card that supports more than one class or to accelerate activation time.

Voltage selection sequence is encoded in one byte as following:

voltageSequence

Sequence			**3**		**2**		**1**	
Bit	7	6	5	4	3	2	1	0
Voltage	0	0	01 – 1.8 V 10 – 3 V 11 – 5 V		01 – 1.8 V 10 – 3 V 11 – 5 V		01 – 1.8 V 10 – 3 V 11 – 5 V	

If all bits are 0, automatic voltage selection is set. That means device driver is responsible for voltage selection. Microsoft CCID driver voltage selection sequence is: 5 V, 3 V, 1.8 V. Bits 7 and 6 are ignored and should be set to 0.

Examples:

1Bh (27dec) = 00011011: 5 V → 3 V → 1.8 V

39h (57dec) = 00111001: 1.8 V → 3 V → 5 V

6.3 Data Exchange Level

CHERRY KC 1000 SC supports following protocols as defined in ISO/IEC 7816 -3 [ISO/IEC 7816-3]:

- T = 0, T = 1
- S = 8, S = 9, S = 10

The data exchange level can be configured in one of the following way:

- TPDU exchange level
- APDU exchange level
- Extended APDU exchange level

exchangeLevel

Value	**Comments**
01h	TPDU exchange level
02h	APDU exchange level
04h	Extended APDU exchange level (only T = 1 cards)

6.3.1 TPDU Exchange Level

Transmission Protocol Data Unit level of communication is a kind of exchange with the host. For TPDU level exchanges, the CCID provides the transportation of host's APDU to the ICC's (Integrated Circuit Card) TPDU. This protocol is described in [ISO 7816-3].

Note: For synchronous cards the T = 0 protocol is emulated.

6.3.2 APDU Exchange Level

Application Protocol Data Unit level of communication is a kind of exchange with the host. For APDU level exchanges, the CCID provides the transportation of host's APDU to the ICC's (Integrated Circuit Card) TPDU. Two APDU levels are defined, short APDU and extended APDU. APDU commands and responses are defined in [ISO 7816-4].

6.3.3 Extended APDU Exchange Level

The extended APDU data exchange is supported for T = 1 cards only. For T = 0 cards the application should use the ENVELOPE command instead.

6.4 ISO and EMVCo Mode

6.4.1 ISO mode

This is default mode of operation for smart card reader. It is designated to operate with [ISO 7816-4] compatible cards and synchronous cards.

6.4.2 EMVCo Mode

The mode is suitable to operate with cards that fulfill EMVCo specification.

Compared to ISO Mode:

Only 5 V power supply

Synchronous cards not supported

operatingMode

Value	Comments
01h	ISO/IEC 7816 mode
02h	EMVCo mode

7 PC/SC

With the CHERRY KC 1000 SC, access contact cards through the framework defined in PC/SC. This makes card integration a snap for any developer who is already familiar with this framework.

The Microsoft® Developer Network (MSDN®) Library contains valuable information and complete documentation of the SCard API within the MSDN Platform SDK.

See http://msdn.microsoft.com/en-us/library/windows/desktop/aa380149(v=vs.85).aspx

7.1 How to Access Smart Card or Reader through PC/SC

The following steps provide a guideline to create your first smart card application using industry standard, PC/SC compliant API function calls. The function definitions provided are taken verbatim from the MSDN Library [MSDNLIB]. For additional descriptions of these and other PC/SC functions provided by the Microsoft Windows PC/SC smart card components, reference the MSDN Library. See http://msdn.microsoft.com/en-us/library/ms953432.aspx.

1. Establish Context

   This step initializes the PC/SC API and allocates all resources necessary for a smart card session. The **`SCardEstablishContext`** function establishes the resource manager context (scope) within which database operations is performed.

```
LONG SCardEstablishContext( IN DWORD dwScope,
                            IN LPCVOID pvReserved1,
                            IN LPCVOID pvReserved2,
                            OUT LPSCARDCONTEXT phContext);
```

2. Get Status Change

   Check the status of the reader for card insertion, removal or availability of the reader. This **`SCardGetStatusChange`** function blocks execution until the current availability of the cards in a specific set of readers **change**. The caller supplies a list of monitored readers and the maximum wait time (in milliseconds) for an action to occur on one of the listed readers.

```
LONG SCardGetStatusChange(     IN SCARDCONTEXT hContext,
IN DWORD dwTimeout,
IN OUT LPSCARD_READERSTATE rgReaderStates,
IN DWORD cReaders);
```

3. List Readers

   To acquire a list of all PC/SC readers use the **`SCardListReaders`** function. Look for **Cherry KC 1000 SC** in the returned list. If multiple Contact Smart Card readers are connected to your system, they will be enumerated.

   Example: Cherry KC 1000 SC 1, and Cherry KC 1000 SC 2.

```
LONG SCardListReaders( IN SCARDCONTEXT hContext,
                       IN LPCTSTR mszGroups,
                       OUT LPTSTR mszReaders,
                       IN OUT LPDWORD pcchReaders);
```

4. Connect

   Connect to the card. The **SCardConnect** function establishes a connection (using a specific resource manager context) between the calling application and a smart card contained by a specific reader. If no card exists in the specified reader, an error is returned.

```
LONG SCardConnect( IN SCARDCONTEXT hContext,
                   IN LPCTSTR szReader,
                   IN DWORD dwShareMode,
                   IN DWORD dwPreferredProtocols,
                   OUT LPSCARDHANDLE phCard,
                   OUT LPDWORD pdwActiveProtocol);
```

5. Exchange Data and Commands with the Card or the reader

   Exchange command and data through APDUs. The **SCardTransmit** function sends a service request to the smart card, expecting to receive data back from the card.

```
LONG SCardTransmit( IN SCARDHANDLE hCard,
                    IN LPCSCARD_I0_REQUEST pioSendPci,
                    IN LPCBYTE pbSendBuffer,
                    IN DWORD cbSendLength,
                    IN OUT LPSCARD_IO_REQUEST pioRecvPci,
                    OUT LPBYTE pbRecvBuffer,
                    IN OUT LPDWORD pcbRecvLength);
```

   Note: The application communicates through **SCardControl()** in environments

   not allowing **SCardTransmit()** without an ICC
   not allowing **SCardTransmit()** for any other reasons
   when developers prefer the application communicate through **SCardControl()**.

   The application retrieves the control code corresponding to **FEATURE_CCID_ESC_COMMAND** (see part 10, rev.2.02.07). In case this feature is not returned, the application may try **SCARD_CTL_CODE** (3500) as control code to use.

```
LONG SCardControl( IN SCARDHANDLE hCard,
                   IN DWORD dwControlCode,
                   IN LPCVOID lpInBuffer,
                   IN DWORD nInBufferSize,
                   OUT LPVOID lpOutBuffer,
                   IN DWORD nOutBufferSize,
                   OUT LPDWORD lpBytesReturned);
```

6. Disconnect

   It is not necessary to disconnect the card after the completion of all transactions, but it is recommended. The **SCardDisconnect** function terminates a connection previously opened between the calling application and a smart card in the target reader.

```
LONG SCardDisconnect( IN SCARDHANDLE hCard,
                      IN DWORD dwDisposition);
```

7. Release

   This step ensures all system resources are released. The **SCardReleaseContext** function closes an established resource manager context, freeing any resources allocated under that context.

```
LONG SCardReleaseContext(IN SCARDCONTEXT hContext);
```

7.2 Vendor Specific Commands

Card reader support features outside the specified commands of PC/SC. To allow applications to control these features a generic command needs to be used. Use of such a generic command prevents conflicts of reserved INS values but used by certain card reader.

This command allows applications to control device specific features provided by the reader.

Command APDU

CLA	INS	P1	P2	Lc	Data Field	Le
FFh	70h	07h	6Bh	xx	DER TLV coded PDU (Vendor Payload)	xx

The IFD supports the INS Byte 70h for vendor specific proprietary commands.

P1 and P2 constitute the vendor ID. For CHERRY KC 1000 SC product is the VID = 076Bh.

The Data Field is constructed as ASN.1 objects/items, whereby every CHERRY KC 1000 SC object is identified by a unique Object Identifier (OID).

OIDs are organized as a leaf tree under an invisible root node. The following table shows the first root nodes.

Vendor Command	Tag Value	Vendor Payload Branch
	A2h (constructed)	readerInformationApi
	BCh (constructed)	deviceSpecificCommand
	9Dh (primitive) BDh (constructed)	response
	9Eh (primitive)	errorResponse

Subchapters present all OIDs.

7.2.1 Response APDU

For all commands encapsulated in generic 70h APDU, the IFD returns response frame constructed as follows.

Data field	SW1 SW2
DER TLV coded Response PDU	See ISO 7816-4

Two last bytes of response frame are always the return code, SW1SW2.

In cases of an ISO 7816 violation, the return code is according to ISO 7816-4 and the data field may be empty.

In cases of positive processing or internal errors, the IFD returns SW1SW2 = 9000 and the data field is encapsulated in the response TAG (9Dh or BDh) or error response TAG (9Eh).

The response includes more than one leaf, depending on the request. Each leaf is encapsulated in the leaf tag.

7.2.2 Error Response

The error response TAG caused by the firmware core is 9Eh (Class Context Specific) + (Primitive) + (1Eh). Length is 2 byte. First byte is the cycle in which the error occurred and the second byte is the exception type.

9E 02 xx yy 90 00	
Value	**Description**
9Eh	Tag = Error Response (0Eh) + (Class Context Specific) + (Primitive)
02h	Len = 2
cycle	Value byte 1: Cycle in which the error is occurred, see Error Cycle
error	Value byte 2: Error code, see Error Code
SW1	90
SW2	00

Error Cycle

First value byte	
Cycle	**Description**
0	HID Proprietary Command APDU
1	HID Proprietary Response APDU
2	HID Read or Write EEPROM Structure
3	RFU
4	RFU
5	RFU

Error Code

Second value byte		
Exception		**Description**
3	03h	NOT_SUPPORTED
4	04h	TLV_NOT_FOUND
5	05h	TLV_MALFORMED
6	06h	ISO_EXCEPTION
11	0Bh	PERSISTENT_TRANSACTION_ERROR
12	0Ch	PERSISTENT_WRITE_ERROR
13	0Dh	OUT_OF_PERSISTENT_MEMORY
15	0Fh	PERSISTENT_MEMORY_OBJECT_NOT_FOUND
17	11h	INVALID_STORE_OPERATION
19	13h	TLV_INVALID_SETLENGTH
20	14h	TLV_INSUFFICIENT_BUFFER
21	15h	DATA_OBJECT_READONLY
31	1F	APPLICATION_EXCEPTION (Destination Node ID mismatch)
42	2Ah	MEDIA_TRANSMIT_EXCEPTION (Destination Node ID mismatch)
43	2Bh	SAM_INSUFFICIENT_MSGHEADER (Secure Channel ID not allowed)
47	2Fh	TLV_INVALID_INDEX
48	30h	SECURITY_STATUS_NOT_SATISFIED
49	31h	TLV_INVALID_VALUE
50	32h	TLV_INVALID_TREE

Second value byte		
Exception		**Description**
64	40h	RANDOM_INVALID
65	41h	OBJECT_NOT_FOUND

8 Asynchronous Card Support

Asynchronous cards contain a CPU or are memory cards that are accessed through [ISO7816-4] compliant framed APDU commands. This type of card supports at least one of the asynchronous protocols T=0 or T=1. No additional libraries or third-party software components are necessary to integrate contact CPU cards.

There is no standard list of APDUs (except specified in [PCSC-3]). Typically card has its own list of unique commands. Consult card specification for full list of supported commands.

8.1 Standard APDU

Standard APDU is application data unit that allows sending maximum 255 bytes of data to the card. It is supported by all [ISO7816-4] compatible cards.

8.1.1 Command APDU definition

Command APDU is sent by the reader to the card. It contains mandatory 4 bytes header and from 0 to 255 bytes of data.

CLA	INS	P1	P2	Lc	Command Data	Le

CLA – instruction class

INS – instruction code

P1, P2 – command parameters

Lc – number of command data bytes to follow

Command data – Lc number bytes of data

Le - maximum number of expected response bytes

If Le is omitted, any number of bytes in response is accepted. If Le=0, reader do not expect any response data (except 2 status bytes SW1, SW2).

If length of command data is 0, Lc must be omitted.

8.1.2 Response APDU

Response APDU is sent by the card to the reader. It contains from 0 to 255 bytes of data and 2 mandatory status bytes SW1 and SW2.

Response Data	SW1	SW2

8.2 Extended APDU

Extended APDU is extension to standard APDU. It allows transmit more than 255 bytes in command and response. Extended APDU is backward compatible with Standard APDU. Command and response APDU looks exactly the same for both types if data length is equal or less to 255 bytes.

To use extended APDU card must support it and reader must operate in Extended APDU mode. Refer to Extended APDU exchange level chapter how to enable this mode.

The only difference between Standard APDU and Extended APDU is length of Lc and Le fields. In Extended APDU these may be omitted, 1 or 3 bytes depending of the length of data.

If length of command data is less or equal to 255 the same rules apply to Lc as for Standard APDU. If length of data is greater than 255, Lc must be 3 bytes length with first byte equal to 0 and two following bytes encoding actual command data length.

Similar rules apply to Le. It may be omitted or 1 byte in the same way as for Standard APDU. If expected more than 255 bytes of data, it may be 2 bytes long if Lc indicates extended length or 3 bytes long with first byte equal to 0 if there is no Lc field or Lc indicates length less than 256 bytes (Standard APDU).

For more detailed description of data length encoding rules, please see [ISO7816-4].

9 Synchronous Cards Support

The device provides two ways to access synchronous cards. One option is PC/SC – like command set that uses standard APDU syntax and standard SCardTransmit() API, but use the reserved value of the CLA byte of ‘FF’. The other is through Vendor Specific proprietary synchronous API.

9.1 PC/SC Command Set

The following functions are supported through PC/SC:

Command	Comments
READ_BINARY	Implemented according to PC/SC [PC/SC spec].
UPDATE_BINARY	Implemented according to PC/SC [PC/SC spec].
VERIFY	Implemented according to PC/SC [PC/SC spec].
READ_PROTECTION_MEMORY	Proprietary extension to PC/SC. Applicable for SLE 4418/28/32/42, SLE 5518/28/32/42.
COMPARE_AND_PROTECT	Proprietary extension to PC/SC. Applicable for SLE 4418/28/32/42, SLE 5518/28/32/42.
MODIFY	Proprietary extension to PC/SC. Applicable for SLE 4418/28/32/42, SLE 5518/28/32/42.
I2C_INIT	Proprietary extension to PC/SC. Applicable for I2C cards.
I2C_WRITE	Proprietary extension to PC/SC. Applicable for I2C cards.
I2C_READ	Proprietary extension to PC/SC. Applicable for I2C cards.

9.1.1 Read Binary

This function reads data from the synchronous smart card. If the Le field is set to ‘00’, then all bytes until the end of the file shall be read within the limit of 256 for a short Le field.

Read Binary Command

CLA	INS	P1	P2	Lc	Data In	Le
FFh	B0h	Address MSB	Address LSB	–	–	xx

Note: For I^2C memory cards with more than 64kB, i.e. AT24C1024SC with 1Mb (128KB) of memory, please use I2C_READ command.

Read Binary Output

Data Out
Data + SW1 SW2

Read Binary Error Codes

	SW1	SW2	Meaning
Warning	62h	81h 82h	Part of returned data may be corrupted. End of file reached before reading expected number of bytes.
Error	69h	81h 82h 86h	Command incompatible. Security status not satisfied (no PIN presented) Command not allowed.
	6Ah	81h 82h	Function not supported. File not found / Addressed block or byte does not exist.
	6Ch	xxh	Wrong length (wrong number Le; 'xx' is the exact number).

9.1.2 Update Binary

This function writes data on the synchronous smart card. If the Lc field is set to '00', then 256 bytes until the end of the file shall be written for a short Lc field.

Update Binary Command

CLA	INS	P1	P2	Lc	Data In	Le
FFh	D6h	Address MSB	Address LSB	xx	Data	–

Note: For I^2C memory cards with more than 64kB, i.e. AT24C1024SC with 1Mb (128KB) of memory, please use I2C_WRITE command.

Update Binary Output

Data Out
SW1 SW2

Update Binary Error Codes

	SW1	SW2	Meaning
Warning	62h	81h 82h	Part of returned data may be corrupted. End of file reached before reading expected number of bytes.
Error	65h	81h	Memory failure (unsuccessful writing).
	69h	81h 82h 86h	Command incompatible. Security status not satisfied (no PIN presented) Command not allowed.
	6Ah	81h 82h	Function not supported. File not found / Addressed block or byte does not exist.

9.1.3 Verify

For some synchronous cards (SLE 4428/42, SLE 5528/42) security code logic exists which controls the write/erase access to the memory.

The SLE 4442/5542 contains a 4-byte security memory with an Error Counter EC (bit 0 to bit 2) and 3 bytes reference data (PIN).

The SLE 4428/5528 contains a 3-byte security memory with Error Counter EC and 2 bytes reference data (PIN).

After power on the whole memory, except for the reference data (PIN), can only be read. Only after a successful comparison of verification data with the internal reference data the memory has full access functionality (read/write/erase) until power is switched off.

The value of the Error Counter determines the number of possible attempts to verify PIN code. For SLE 4428/5528 there are maximum 8 tries, for SLE 4442/5542 there are maximum 3 tries.

Verify Command

CLA	INS	P1	P2	Lc	Data In	Le
FFh	20h	00h	00h	PIN length	PIN	–

NOTE: Update Binary Extended Command with first byte of the Le-Field equals to 0x00 is not supported.

Verify Output

Data Out
SW1 SW2

Verify Error Codes

	SW1	**SW2**	**Meaning**
Error	65h	81h	Memory failure (unsuccessful writing).
	69h	82h 83h 84h	Security status not satisfied. Verify method blocked. Reference data not usable.
	6Ah	88h	Unknown / not supported Card-Protocol.

9.1.4 Read Protection Memory

For each byte of some synchronous cards (SLE 4418/28/32/42, SLE 5518/28/32/42) a protection bit exists. This bit should be retrievable for applications.

Read Protection Memory Command

CLA	INS	P1	P2	Lc	Data In	Le
FFh	3Ah	Address MSB	Address LSB	–	–	xx

Each bit of the protection memory is returned in one complete byte and occupies bit 0. Bit 1–7 are RFU and set 0x00.

For the sequence of the protection bits LSB coding is used.

NOTE: Read Protection Memory Extended Command with first byte of the Le-Field equals to 0x00 is not supported.

Read Protection Memory Output

Data Out
Data + SW1 SW2

0x00: Protection bit is not set
0x01: Protection bit is set

Read Protection Memory Error Codes

	SW1	SW2	Meaning
Warning	62h	82h	End of file reached before reading expected number of bytes.
Error	69h	81h 82h 86h	Command incompatible. Security status not satisfied (no PIN presented) Command not allowed.
	6Ah	81h 82h	Function not supported. File not found / Addressed block or byte does not exist.
	6Fh	00h	Unknown / not supported Card-Protocol.

9.1.5 Compare and Protect

For each byte of some synchronous cards (SLE 4418/28/32/42, SLE 5518/28/32/42) a protection bit exists. This command compares the data with the memory and set the protection bit if the data matches with the memory and the protection bit is not set for this data byte.

SLE 4432/42, SLE 5532/42 have a special protection memory. In this case the function only compares data in the protection memory.

Compare and Protect Command

CLA	INS	P1	P2	Lc	Data In	Le
FFh	30h	00h	03h	5 + N	See Data In – field buildup table below	–

NOTE: Compare and Protect Extended Command with first byte of the Lc-Field equals to 0x00 is not supported.

Data In - field buildup

Version	Flag 1	Flag 2	Address	Address	Data to Compare
01h	00h	00h	MSB	LSB	N Data-Bytes

If the current data byte matches the memory, the protection set will be set. If an error occurs by one of the following data bytes, all protection bits before are set.

Compare and protect Output

Data Out
(Address MSB LSB) + SW1 SW2

The address is only returned if an error occurred while comparing the data bytes.

Compare and Protect Error Codes

	SW1	SW2	Meaning
Warning	62h	82h	End of file reached before reading expected number of bytes.
Error	69h	81h 82h 86h	Command incompatible. Security status not satisfied (no PIN presented) Command not allowed.
	6Ah	81h 82h	Function not supported. File not found / Addressed block or byte does not exist.
	6Fh	00h	Unknown / not supported Card-Protocol.

9.1.6 Modify

Synchronous cards like SLE4418/28/32/42, SLE5518/28/32/42 provide a functionality to change the current PIN.

Modify Command

CLA	INS	P1	P2	Lc	Data In	Le
FFh	21h	00h	00h	PIN length	Old PIN + New PIN	–

Note: Old PIN + New PIN always needed for correct Modify execution.

Modify Output

Data Out
SW1 SW2

Modify Error Codes

	SW1	SW2	Meaning
Warning	63h	00h CXh	No information is given. Counter (verification failed; 'X' encodes the number of further allowed retries).
Error	65h	81h	Memory failure (unsuccessful writing).
	69h	82h 83h	Security status not satisfied. Verify method blocked.
	6Fh	00h	Unknown / not supported Card-Protocol.

9.1.7 I2C Init

This command initializes an I2C card with parameters like Page size, Number of Address Bytes and Memory size. Predefined types of cards can be called with the Type - byte. If the Type - byte is zero, the own parameters will be taken if they are valid.

I2C Init Command

CLA	INS	P1	P2	Lc	Data In	Le
FFh	30h	00h	04h	08h	See Data In – field buildup table below	–

Data In - field buildup

Version (1 Byte)	Type (1 Byte)	Page Size (1 Byte)	Number of Address Bytes (1 Byte)	Memory Size (4 Bytes, MSB first)
01h	See Type - definitions table below			

Type - definitions

Type	Page Size	Number of Address Bytes	Memory Size	Card
00h	XX	XX	MSB XX XX LSB	undefined
01h	8	1	256	ST14C02C
02h	8	1	512	ST14C04C
03h	32	2	4096	ST14E32
04h	16	1	512	M14C04
05h	16	1	2048	M14C16
06h	32	2	4096	M14C32
07h	32	2	8192	M14C64
08h	64	2	16384	M14128
09h	64	2	32768	M14256
0Ah	8	1	256	GFM2K
0Bh	16	1	512	GFM4K
0Ch	32	2	4096	GFM32K
0Dh	8	1	128	AT24C01A
0Eh	8	1	256	AT24C02
0Fh	16	1	512	AT24C04
10h	16	1	1024	AT24C08
11h	16	1	2048	AT24C16
12h	16	1	2048	AT24C164
13h	32	2	4096	AT24C32
14h	32	2	8192	AT24C64
15h	64	2	16384	AT24C128
16h	64	2	32768	AT24C256
17h	64	2	16384	AT24CS128
18h	64	2	32768	AT24CS256
19h	128	2	65536	AT24C512
1Ah	0 (=256)	2	131072	AT24C1024
1Bh	4	1	256	X24026

I2C Init Output

Data Out
SW1 SW2

I2C Init Error Codes

	SW1	SW2	Meaning
Warning	63h	00h	No information is given.
Error	69h	86h	Command not allowed (wrong parameters).
	6Ah	81h	Function not supported.
	6Fh	00h	Unknown / not supported Card-Protocol.

9.1.8 I2C Write

This command writes data on the I2C card. This command can only write to a maximum of 250 Bytes at once.

I2C Write Command

CLA	INS	P1	P2	Lc	Data In	Le
FFh	30h	00h	06h	5 + N	See Data In – field buildup table below	–

Lc-Field contains how many Data-Bytes you want to write. 5 Bytes need to be added because of the 5 Byte-Header.

Note: I2C Write Extended Command with first byte of the Lc-Field equals to 0x00 is not supported.

Data In - field buildup

Version (1 Byte)	Address Bytes (4 Bytes, MSB first)	Data (0 – 250 Bytes)
01h	MSB XX XX LSB	Data to write

I2C Write Output

Data Out
SW1 SW2

I2C Write Error Codes

	SW1	SW2	Meaning
Warning	62h	82h	End of file reached before reading expected number of bytes.
Error	69h	81h 86h	Command incompatible. Command not allowed.
	6Ah	81h 82h	Function not supported. File not found / Addressed block or byte does not exist.
	6Fh	00h	Unknown Protocol / Memory Allocation failure.

9.1.9 I2C Read

This command reads the content from the I2C card.

I2C Read Command

CLA	INS	P1	P2	Lc	Data In	Le
FFh	30h	00h	05h	09h	See Data In – field buildup table below	–

Data In-Field buildup

Version (1 Byte)	Address Bytes (4 Bytes, MSB first)	Data (0 – 250 Bytes)
01h	MSB XX XX LSB	Data to write

I2C Read Output

Data Out
SW1 SW2

I2C Read Error Codes

	SW1	SW2	Meaning
Warning	62h	82h	End of file reached before reading expected number of bytes.
Error	69h	81h 86h	Command incompatible. Command not allowed.
	6Ah	81h 82h	Function not supported. File not found / Addressed block or byte does not exist.
	6Fh	00h	Unknown Protocol / Memory Allocation failure.

9.2 Vendor Specific Synchronous Command Set

These commands allow applications to communicate with Synchronous Contact Cards using raw native card commands directly sent to the card by proper Bus Protocol.

All Synchronous Contact Card commands are identified by unique ASN.1 leaf. The root is defined as Synchronous Card Command and is encapsulated in vendor specific generic command see Table 30. Under this root are specific commands for 2WBP (2 Wire Bus Protocol), 3WBP (3 Wire Bus Protocol) and I2C cards, organized as follows.

Synchronous Card Command Structure

Vendor Command	Tag Value	Native Card Command
	A6h (constructed)	2WBPReaderWrite [A0h] 3WBPReaderWrite [A1h] I2CReaderWrite [A2h]

9.2.1 2WBP Read/Write

This command allows communication with synchronous contact smart card which supports 2 Wire Bus Protocol (2WBP) such as SLE 4432/42.

Each command consists of three bytes:

- Control
- Address
- Data

For SLE 4432 there are available 4 commands which are listed in table below.

2WBP Common Commands

Byte 1 Control								Byte2 Address	Byte 3 Data	Operation
B7	B6	B5	B4	B3	B2	B1	B0	A7 - A0	D7 - D0	
0	0	1	1	0	0	0	0	address	no effect	Read Main Memory
0	0	1	1	1	0	0	0	address	input data	Update Main Memory
0	0	1	1	0	1	0	0	no effect	no effect	Read Protection Memory
0	0	1	1	1	1	0	0	address	input data	Write Protection Memory

Additionally to above commands there are available 3 commands for the SLE 4442 which can be found in table below.

2WBP SLE 4442 only Commands

Byte 1 Control								Byte2 Address	Byte 3 Data	Operation
B7	B6	B5	B4	B3	B2	B1	B0	A7 - A0	D7 - D0	
0	0	1	1	0	0	0	1	no effect	no effect	Read Security Memory
0	0	1	1	1	0	0	1	address	input data	Update Security Memory
0	0	1	1	0	0	1	1	address	input data	Compare Verification Data

More information can be found in “ICs for Chip Cards Intelligent 256-Byte EEPROM SLE 4432/SLE 4442” datasheet prepared by SIEMENS.

Example Update Main Memory:

command:

```
FF 70 07 6B 07                  // Vendor Specific APDU with 7 bytes object
    A6 05                       // Synchronous Card Command
        A0 03                   // 2WBPReadWrite
            38 AA 55            // Control = 0x38, Address = 0xAA, Data = 0x55
                  00            // Le
```

reply:

```
BD 02                           // Response
    A0 00                       // Received data (0 bytes)
        90 00                   // SW1 SW2
```

Example Read Main Memory:

command:

```
FF 70 07 6B 07                  // Vendor Specific APDU with 7 bytes object
    A6 05                       // Synchronous Card Command
        A0 03                   // 2WBPReadWrite
            30 AA 00            // Control = 0x30, Address = 0xAA, Data = 0x00
                  00            // Le
```

reply:

```
BD 03                           // Response
    A0 01                       // Received data (1 byte)
        55                      // Data
            90 00               // SW1 SW2
```

9.2.2 3WBP Read/Write

This command allows communication with synchronous contact smart card which supports 3 Wire Bus Protocol (3WBP) such as SLE 4418/28.

Each command consists of three bytes.

3WBP Control Words for Command Entry

Byte 1								Byte2	Byte 3	Operation
A9	A8	S5	S4	S3	S2	S1	S0	A7 - A0	D7 - D0	
address bit 9 and 0		1	1	0	0	0	1	address Bit 7 - 0	input data	Write and erase with protect bit
		1	1	0	0	1	1		input data	Write and erase without protect bit
		1	1	0	0	0	0		comparison data	Write protect bit with data comparison (verification)
		0	0	1	1	0	0		no effect	Read 9 bits, data with protect bit
		0	0	1	1	1	0		no effect	Read 8 bits, data without protect bit
		1	1	0	0	0	1		input data	Write and erase with protect bit

3WBP Control Words for Command Entry, User Identification

Byte 1								Byte2	Byte 3	Operation
A9	A8	S5	S4	S3	S2	S1	S0	A7 - A0	D7 - D0	
1	1	1	1	0	0	1	0	0xFD	bit mask	Write error counter
1	1	0	0	1	1	0	1	0xFE	PIN byte 1	Verify 1st PIN byte
1	1	0	0	1	1	0	1	0xFF	PIN byte 2	Verify 2nd PIN byte

More information can be found in “ICs for Chip Cards SLE 4418/SLE 4428 Intelligent 8-Kbit EEPROM” datasheet prepared by SIEMENS.

Example Read Memory with protect bit:

command:

```
FF 70 07 6B 07                  // Vendor Specific APDU with 7 bytes object
    A6 05                       // Synchronous Card Command
        A1 03                   // 3WBPReadWrite
            0C AA 00            // address = 0xAA
                00              // Le
reply:
BD 04                           // Response
    A1 02                       // Received data (2 bytes)
        AA 80                   // Data = 0xAA, Protect bit = 0x80 (set)
            90 00               // SW1 SW2
```

Example Write Memory without protect bit:

command:

```
FF 70 07 6B 07                  // Vendor Specific APDU with 7 bytes object
    A6 05                       // Synchronous Card Command
        A1 03                   // 3WBPReadWrite
            33 AA BB            // address = 0xAA, data = 0xBB
                00              // Le
reply:
BD 02                           // Response
    A1 00                       // Received data (0 bytes)
        90 00                   // SW1 SW2
```

9.2.3 I2C Read/Write

This command allows communication with synchronous contact smart card which supports I2C Bus Protocol such as AT24C01/02/04/…/1024 etc.

Each command consists of 5+N bytes.

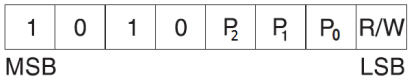
I2C Read/Write Command

Offset	Field	Size	Value	Description
0	Address length	1	1,2,3	length of address: 1… slave address only 2… slave address + byte-subaddress 3… slave address + word-subaddress
1	Bytes to read/write	1	N	# of bytes to read/write
2	Address	1		device address
3	Subaddress 1	1		1st byte of subaddress
4	Subaddress 2	1		2nd byte of subaddress
5	Data	N		array of data bytes

NOTE: 32 bytes is the maximum value of “Bytes to read/write” field.

The device address word consists of a mandatory one, zero sequence for the first four most significant bits as shown. This is common to all I2C EEPROM devices.

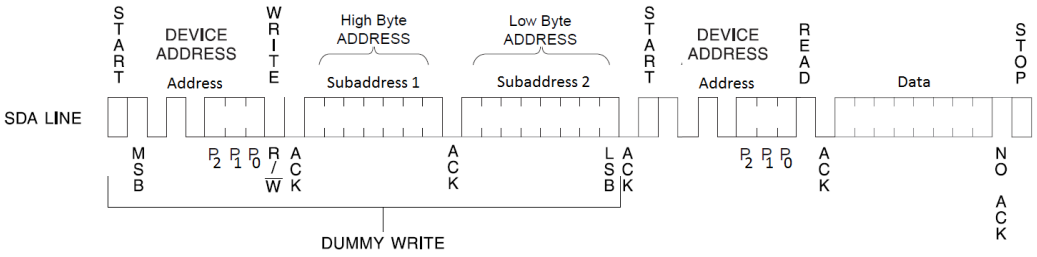
I2C Device Address



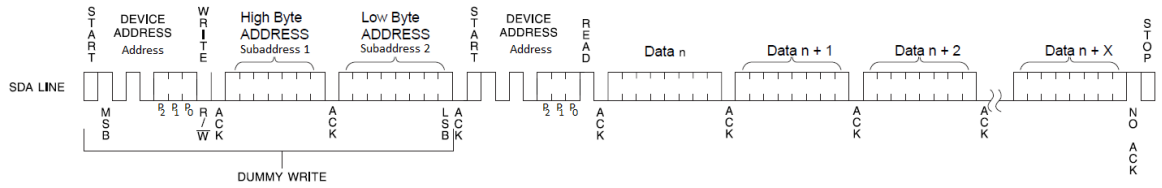
The next 3 bits are used for memory page addressing. These page addressing bits should be considered the most significant bits of the data word address.

More information about I2C EEPROM memory addressing, can be found in datasheets of “Two-wire serial EEPROM” prepared by Atmel, e.g. AT24C01 or AT24C1024.

I2C Byte Read Command

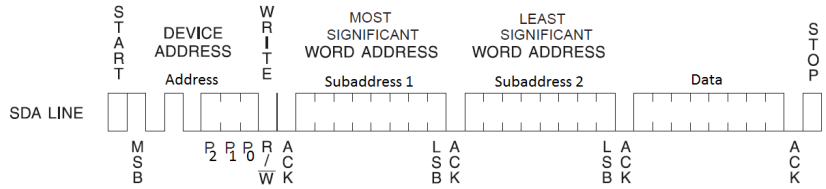


I2C N-Bytes Read Command

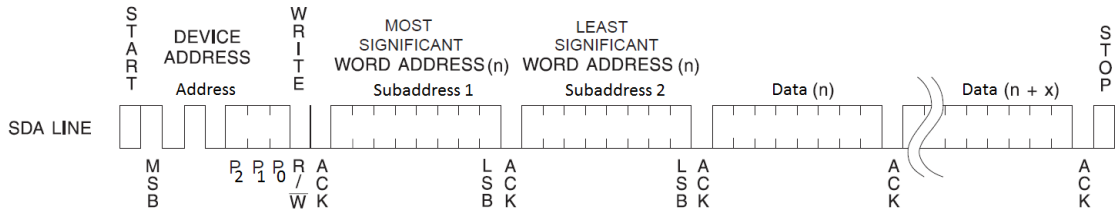


It must be checked in proper synchronous contact smart card documentation to check how big address length is. For some Smart Cards such as AT24C01 2 bytes subaddress is not necessary.

I2C Byte Write Command



I2C N-Bytes Write Command



N-Byte Write command can write maximum 32 bytes at once, but when word address internally generated (inside the Smart Card EEPROM memory), reaches the page boundary, the following byte is placed at the beginning of the same page. The address “rollover” during write is from the last byte of the current page to the first byte of the same page.

It must be checked in proper synchronous contact smart card documentation to check how big the memory page is.

Example Read 16 bytes, address 0x0100 (AT24C1024, 3 bytes of address length):

command:

```
FF 70 07 6B 09                  // Vendor Specific APDU with 9 bytes object
    A6 07                       // Synchronous Card Command
        A2 05                   // I2CReadWrite
            03 10 A1 01 00      // 5 bytes of command
                00              // Le
```

reply:

```
BD 12                           // Response
    A2 10                       // Received data (16 bytes)
        16 Bytes of received data   // Data
            90 00               // SW1 SW2
```

Example Read 8 bytes, address 0x50 (AT24C01, 2 bytes of address length):

command:

```
FF 70 07 6B 09                  // Vendor Specific APDU with 9 bytes object
    A6 07                       // Synchronous Card Command
        A2 05                   // I2CReadWrite
            02 08 A1 50 00      // 5 bytes of command
                00              // Le
```

reply:

```
BD 0A                           // Response
    A2 08                       // Received data (8 bytes)
        8 Bytes of received data // Data
            90 00               // SW1 SW2
```

Example Read 1 byte, current address (1 byte of address length):

command:

```
FF 70 07 6B 09              // Vendor Specific APDU with 9 bytes object
    A6 07                   // Synchronous Card Command
        A2 05               // I2CReadWrite
            01 01 A1 00 00  // 5 bytes of command
                00          // Le
```

reply:

```
BD 03                                   // Response
    A2 01                               // Received data (1 bytes)
        XX                              // Data
            90 00                       // SW1 SW2
```

Example Write 8 bytes, address 0x0100 (AT24C1024, 3 bytes of address length):

command:

```
FF 70 07 6B 11                          // Vendor Specific APDU with 7 bytes object
  A6 0F                                 // Synchronous Card Command
    A2 0D                               // I2CReadWrite
      03 08 A0 01 00                    // 5 bytes of command
        XX XX XX XX XX XX XX XX         // Data to write
          00                            // Le
```

reply:

```
BD 02                                   // Response
    A2 00                               // Received data (0 bytes)
        90 00                           // SW1 SW2
```

10 Secure Pin Entry Support

CHERRY KC 1000 SC implements a method of secure pin entry called SPE. This feature uses PCSC specification: Part 10. IFDs with Secure Pin Entry Capabilities [PCSC-10-SPE].

10.1 Feature Execution by Pseudo-APDU

SPE is commanded by special APDU's, called Pseudo APDU (PPDU). The Pseudo-APDU command is in a data format which has much resemblance with an APDU for cards:

CLA	INS	P1	P2	Lc	Data Field	Le
FFh	C2h	01h	*Feature Number*	Lc	*Feature Command Data*	Le

Any valid Pseudo-APDU command will always generate a response:

Response Data	Response Status SW1/SW2	Description
Feature Response Data	90 00	Feature executed successful, Feature Response Data is present
-empty-	6A 86	Incorrect value for P2 (requested feature not present)

The list of Feature Numbers and Feature Command Data implemented by the CHERRY KC 1000 SC is defined in the following table:

Feature	Feature Number	Feature Command Data	Feature Response Data
GET_FEATURE_REQUEST	0x00	-empty-	byte array: each byte in this array represents a feature number present in this reader
FEATURE_VERIFY_PIN_START	0x01	PIN_VERIFY structure (see below)	-empty-
FEATURE_VERIFY_PIN_FINISH	0x02	-empty-	2-byte status
FEATURE_MODIFY_PIN_START	0x03	PIN_MODIFY structure (see below)	-empty-
FEATURE_MODIFY_PIN_FINISH	0x04	-empty-	2-byte status
FEATURE_GET_KEY_PRESSED	0x05	-empty-	single byte with the following code: 0x2B – 0-9 valid key 0x1B – cancel 0x08 – backspace 0x0d – enter 0x0e – timeout 0x00 – no key 0x40 – aborted
FEATURE_VERIFY_PIN_DIRECT	0x06	PIN_VERIFY structure (see below)	2-byte status
FEATURE_MODIFY_PIN_DIRECT	0x07	PIN_MODIFY structure (see below)	2-byte status
FEATURE_IFD_PIN_PROPERTIES	0x0A	-empty-	PIN_PROPERTIES

Feature	Feature Number	Feature Command Data	Feature Response Data
			structure (see below)
FEATURE_GET_KEY	0x10	GET_KEY structure (see below)	single byte with the following code: 0x30 – 0 0x31 – 1 0x32 – 2 0x33 – 3 0x34 – 4 0x35 – 5 0x36 – 6 0x37 – 7 0x38 – 8 0x39 – 9 0x2A – * 0x2E – . (dot) 0x1B – cancel 0x08 – backspace 0x0D – enter

PIN_VERIFY structure as defined in PCSC Part 10 specification (paragraph 2.5.2)

Byte Offset	Field	Type	Description
0	bTimeOut	BYTE	timeout in seconds (00 means use default timeout)
1	bTimeOut2	BYTE	timeout in seconds after first key stroke
2	bmFormatString	BYTE	formatting options USB_CCID_PIN_FORMAT_xxx
3	bmPINBlockString	BYTE	bits 7-4 bit size of PIN length in APDU bits 3-0 PIN block size in bytes after justification and formatting
4	bmPINLengthFormat	BYTE	bits 7-5 RFU, bit 4 set if system units are bytes clear if system units are bits, bits 3-0 PIN length position in system units
5	wPINMaxExtraDigit	USHORT	XXYY, where XX is minimum PIN size in digits, YY is maximum
7	bEntryValidationCondition	BYTE	Conditions under which PIN entry should be considered complete: 01h Max size reached 02h Validation key pressed 04h Timeout occurred
8	bNumberMessage	BYTE	Number of messages to display for PIN verification
9	wLangId	USHORT	Language for messages
11	bMsgIndex	BYTE	Message index (should be 00)
12	bTeoPrologue	BYTE[3]	T=1 I-block prologue field to use (fill with 00)
15	ulDataLength	ULONG	length of Data to be sent to the ICC
19	abData	BYTE[]	Data to send to the ICC

PIN_MODIFY structure as defined in PCSC Part 10 specification (paragraph 2.5.3)

Byte Offset	Field	Type	Description
0	bTimeOut	BYTE	timeout in seconds (00 means use default timeout)
1	bTimeOut2	BYTE	timeout in seconds after first key stroke
2	bmFormatString	BYTE	formatting options USB_CCID_PIN_FORMAT_xxx
3	bmPINBlockString	BYTE	bits 7-4 bit size of PIN length in APDU bits 3-0 PIN block size in bytes after justification and formatting
4	bmPINLengthFormat	BYTE	bits 7-5 RFU, bit 4 set if system units are bytes clear if system units are bits, bits 3-0 PIN length position in system units
5	bInsertionOffsetOld	BYTE	Insertion position offset in bytes for the current PIN
6	bInsertionOffsetNew	BYTE	Insertion position offset in bytes for the new PIN
7	wPINMaxExtraDigit	USHORT	XXYY, where XX is minimum PIN size in digits, YY is maximum
9	bConfirmPIN	BYTE	Flags governing need for confirmation of new PIN
10	bEntryValidationCondition	BYTE	Conditions under which PIN entry should be considered complete: 01h Max size reached 02h Validation key pressed 04h Timeout occurred
11	bNumberMessage	BYTE	Number of messages to display for PIN verification
12	wLangId	USHORT	Language for messages
14	bMsgIndex1	BYTE	Index of 1st prompting message
15	bMsgIndex2	BYTE	Index of 2nd prompting message
16	bMsgIndex3	BYTE	Index of 3rd prompting message
17	bTeoPrologue	BYTE[3]	T=1 I-block prologue field to use (fill with 00)
20	ulDataLength	ULONG	length of Data to be sent to the ICC
24	abData	BYTE[]	Data to send to the ICC

PIN_PROPERTIES structure as defined in PCSC Part 10 specification (paragraph 2.5.5)

Byte Offset	Field	Type	Description
0	wLcdLayout	USHORT	Display characteristics 0000h = No LCD
2	bEntryValidationCondition	BYTE	Conditions under which PIN entry should be considered complete: 01h = Max size reached 02h = Validation key pressed 04h = Timeout occurred
3	bTimeOut2	BYTE	0 = IFD does not distinguish bTimeOut from bTimeOut2 1 = IFD distinguishes bTimeOut from bTimeOut2

GET_KEY structure as defined in PCSC Part 10 specification (paragraph 2.5.11)

Byte Offset	Field	Type	Description
0	wWaitTime	USHORT	Time in seconds to wait for a key to be hit
2	bMode	BYTE	Display mode of key N/A, set 00h
3	bPosX	BYTE	Column (starting at 0) N/A, set 00h
4	bPosY	BYTE	Row (starting at 0) N/A, set 00h

10.2 SPE Examples

10.2.1 Get Feature Request

P-APDU (GET_FEATURE_REQUEST):

CLA	INS	P1	P2	Lc	Data Field	Le
FFh	C2h	01h	0x00	0x00	*-empty-*	*-empty-*

Response:

Response Data	Response Status SW1/SW2	Description
01 02 03 04 05 06 07 0A 10	90 00	The list of features

10.2.2 Get PIN_PROPERTIES Structure

P-APDU (FEATURE_IFD_PIN_PROPERTIES):

CLA	INS	P1	P2	Lc	Data Field	Le
FF	C2	01	0A	00	*-empty-*	*-empty-*

Response:

Response Data	Response Status SW1/SW2	Description
00 00 07 01	90 00	PIN_PROPERTIES structure: wLcdLayout = 0x0000 (no display supported) bEntryValidationCondition = 0x07 (all validation condition supported) bTimeOut2 = 0x01 (IFD distinguishes bTimeOut from bTimeOut2)

10.2.3 Direct PIN Verification

P-APDU (FEATURE_VERIFY_PIN_DIRECT):

CLA	INS	P1	P2	Lc	Data Field	Le
FF	C2	01	06	21	0A 05 46 08 00 08 04 06 FF 00 00 00 00 00 00 0E 00 00 00 00 20 00 00 09 FF FF FF FF FF FF FF FF FF PIN_VERIFY structure: bTimeOut=0A bTimeOut2=05 bmFormatString=46 bmPINBlockString=08 bmPINLengthFormat=00 wPINMaxExtraDigit=0408 bEntryValidationCondition=06 bNumberMessage=FF wLangId=0000 bMsgIndex=00 bTeoPrologue=000000 ulDataLength=0000000E abData=0020000009FFFFFFFFFFFFFFFFF	00

Response:

Response Data	Response Status SW1/SW2	Description
90 00	90 00	PIN verification ok

10.2.4 Direct PIN Modification

P-APDU (FEATURE_MODIFY_PIN_DIRECT):

CLA	INS	P1	P2	Lc	Data Field	Le
FF	C2	01	07	29	0A 0A 01 0C 00 00 00 04 04 00 02 FF 00 00 00 00 00 00 00 00 11 00 00 00 00 20 00 00 0C FF FF FF FF FF FF FF FF FF FF FF FF PIN_MODIFY structure: bTimeOut=0A bTimeOut2=0A bmFormatString=01 bmPINBlockString=0C bmPINLengthFormat=00 bInsertionOffsetOld=00 bInsertionOffsetNew=00 wPINMaxExtraDigit=0404 bConfirmPIN=00 bEntryValidationCondition=02 bNumberMessage=FF wLangId=0000 bMsgIndex1=00 bMsgIndex2=00 bMsgIndex3=00 bTeoPrologue[3]=000000 ulDataLength=00000011 abData=002000000CFFFFFFFFFFFFFFFFFFFFFFFF	00

Response:

Response Data	Response Status SW1/SW2	Description
90 00	90 00	PIN modification ok

10.2.5 Indirect PIN Verification

P-APDU (FEATURE_VERIFY_PIN_START):

CLA	INS	P1	P2	Lc	Data Field	Le
FF	C2	01	01	21	0A 05 46 08 00 08 04 06 FF 00 00 00 00 00 00 0E 00 00 00 00 20 00 00 09 FF FF FF FF FF FF FF FF FF PIN_VERIFY structure: bTimeOut=0A bTimeOut2=05 bmFormatString=46 bmPINBlockString=08 bmPINLengthFormat=00 wPINMaxExtraDigit=0408 bEntryValidationCondition=06 bNumberMessage=FF wLangId=0000 bMsgIndex=00 bTeoPrologue=000000 ulDataLength=0000000E abData=0020000009FFFFFFFFFFFFFFFFF	00

Response:

Response Data	Response Status SW1/SW2	Description
-empty-	90 00	

P-APDU (FEATURE_GET_KEY_PRESSED):

This P-APDU should be repeated until one of the termination codes has been returned

CLA	INS	P1	P2	Lc	Data Field	Le
FF	C2	01	01	21	-empty-	-empty-

Response:

Response Data	Response Status SW1/SW2	Description
2B	90 00	Valid key pressed

P-APDU (FEATURE_VERIFY_PIN_FINISH):

CLA	INS	P1	P2	Lc	Data Field	Le
FF	C2	01	02	00	-empty-	-empty-

Response:

Response Data	Response Status SW1/SW2	Description
90 00	90 00	PIN verification ok

10.2.6 Indirect PIN Modification

P-APDU (FEATURE_MODIFY_PIN_START):

CLA	INS	P1	P2	Lc	Data Field	Le
FF	C2	01	03	29	0A 0A 01 0C 00 00 00 04 04 00 02 FF 00 00 00 00 00 00 00 00 11 00 00 00 00 20 00 00 0C FF FF FF FF FF FF FF FF FF FF FF FF PIN_MODIFY structure: bTimeOut=0A bTimeOut2=0A bmFormatString=01 bmPINBlockString=0C bmPINLengthFormat=00 bInsertionOffsetOld=00 bInsertionOffsetNew=00 wPINMaxExtraDigit=0404 bConfirmPIN=00 bEntryValidationCondition=02 bNumberMessage=FF wLangId=0000 bMsgIndex1=00 bMsgIndex2=00 bMsgIndex3=00 bTeoPrologue[3]=000000 ulDataLength=00000011 abData=002000000CFFFFFFFFFFFFFFFFFFFFFFFF	00

Response:

Response Data	Response Status SW1/SW2	Description
-empty-	90 00	

P-APDU (FEATURE_GET_KEY_PRESSED):

This P-APDU should be repeated until one of the termination codes has been returned

CLA	INS	P1	P2	Lc	Data Field	Le
FF	C2	01	01	21	-empty-	-empty-

Response:

Response Data	Response Status SW1/SW2	Description
2B	90 00	Valid key pressed

P-APDU (FEATURE_MODIFY_PIN_FINISH):

CLA	INS	P1	P2	Lc	Data Field	Le
FF	C2	01	04	00	-empty-	-empty-

Response:

Response Data	Response Status SW1/SW2	Description
90 00	90 00	PIN verification ok

11 Device Configuration

All CHERRY KC 1000 SC configuration items are identified by a unique ASN.1 leaf. The root is defined as Reader Information API and is encapsulated in a proprietary command described in 7.2 Vendor Specific Commands, page 16.

The root tag **readerInformationApi** A2h is reserved for GET and SET of reader specific information and provides access to reader configuration.

For a Reader Information GET requests the Le byte must be present, and the Response Tag (1D) is always CONSTRUCTED.

Under the root and the get/set request are a number of branches, organized as follows:

Reader Information Structure

Vendor Command	Reader Information API	Request	ASN1 name of Branches
FF 70 07 6B Lc	Tag = A2h	Get [A0h] Set [A1h]	readerCapabilities [A0h] (read only) tlvVersion[80h] deviceID[81h] productName [82h] productPlatform [83h] firmwareVersion[85h] hardwareVersion[89h] hostInterfaces[8Ah] numberOfContactSlots[8Bh] numberOfContactlessSlots[8Ch] numberOfAntennas[8Dh] vendorName[8Fh] serialNumber[92h] sizeOfUserEEProm[94h] firmwareLabel[96]
			contactSlotConfiguration [A3h] exchangeLevel [80h] voltageSequence [82h] operatingMode [83h]
			readerEEPROM [A7h] eepromOffset [81h] eepromRdLength [82h] eepromWrData [83h]
			readerConfigurationControl [A9h] (write only) rebootDevice [80h]

Note: After SET requests, restart the reader to apply the changes.

11.1 Reader Capabilities

Tag **readerCapabilities A0h** is constructed. One or more primitive sub tags must follow.

Tag	ASN.1 name	Value	Type	Len	Access
TLV Version					
80h	tlvVersion	01h	Uint8_t	1	RO
Get APDU: FF 70 07 6B 08 **A2 06 A0 04 A0 02 80 00** 00 **Response:** BD 03 80 01 01 90 00					
Device ID					
81h	deviceID	00h 03h	Octet String	2	RO
Get APDU: FF 70 07 6B 08 **A2 06 A0 04 A0 02 81 00** 00 **Response:** BD 04 81 02 00 03 90 00					
Name of product					
82h	productName	KC 1000 SC	Null String	11	RO
Get APDU: FF 70 07 6B 08 **A2 06 A0 04 A0 02 82 00** 00 **Response:** BD 0D 82 0B 4B 43 20 31 30 30 30 20 53 43 00 90 00					
Name of processor platform					
83h	productPlatform	AViatoR	Null String	8	RO
Get APDU: FF 70 07 6B 08 A2 06 A0 04 A0 02 83 00 00 **Response:** BD 0A 83 08 41 56 69 61 74 6F 52 00 90 00					
FwVersionMajor + FwVersionMinor + RevisionNr					
85h	firmwareVersion	01h 00h 01h	Octet String	3	RO
Get APDU: FF 70 07 6B 08 **A2 06 A0 04 A0 02 85 00** 00 **Response:** BD 05 85 03 01 00 01 90 00					
Hardware Version					
89h	hardwareVersion	PCB-00062 REV A	Null String	16	RO
Get APDU: FF 70 07 6B 08 **A2 06 A0 04 A0 02 89 00** 00 **Response:** BD 12 89 10 50 43 42 2D 30 30 30 36 32 20 52 45 56 20 41 00 90 00					
Available host communication interfaces, bit 1 = USB					
89h	hostInterfaces	02h	Null String	1	RO
Get APDU: FF 70 07 6B 08 **A2 06 A0 04 A0 02 8A 00** 00 **Response:** BD 03 8A 01 02 90 00					
Number of available contact slots					
8Bh	numberOfContactSlots	01h	Uint8_t	1	RO
Get APDU: FF 70 07 6B 08 **A2 06 A0 04 A0 02 8B 00** 00 **Response:** BD 03 8B 01 01 90 00					
Number of available contactless slots					
8Bh	numberOfContactlessSlots	00h	Uint8_t	1	RO
Get APDU: FF 70 07 6B 08 **A2 06 A0 04 A0 02 8C 00** 00 **Response:** BD 03 8C 01 00 90 00					

Tag	ASN.1 name	Value	Type	Len	Access
Number of available antennas					
8Dh	numberOfAntennas	00h	Uint8_t	1	RO
Get APDU: FF 70 07 6B 08 **A2 06 A0 04 A0 02 8D 00** 00 **Response:** BD 03 8D 01 00 90 00					
Vendor name					
8Fh	vendorName	Cherry	Null String	7	RO
Get APDU: FF 70 07 6B 08 **A2 06 A0 04 A0 02 8F 00** 00 **Response:** BD 09 8F 07 43 68 65 72 72 79 00 90 00					
Serial number					
92h	serialNumber	(empty number)	Octet String	0	RO
Get APDU: FF 70 07 6B 08 **A2 06 A0 04 A0 02 92 00** 00 **Response:** BD 12 92 00 90 00					
Size of user EEProm					
94h	sizeOfUserEEProm	04h 00h	Octet String	2	RO
Get APDU: FF 70 07 6B 08 **A2 06 A0 04 A0 02 94 00** 00 **Response:** BD 12 94 02 04 00 90 00					
Firmware build information					
96h	firmwareLabel	CHKBD-1.0.1.204-20141016T083939-F87D65BE0E1B-RELEASE	Octet String	52	RO
Get APDU: FF 70 07 6B 08 **A2 06 A0 04 A0 02 96 00** 00 **Response:** BD 36 96 34 43 48 4B 42 44 2D 31 2E 30 2E 31 2E 32 30 34 2D 32 30 31 34 31 30 31 36 54 30 38 33 39 33 39 2D 46 38 37 44 36 35 42 45 30 45 31 42 2D 52 45 4C 45 41 53 45 90 00					

11.2 Contact Slot Configuration

The following tags can be used to configure contact slot in **`contactSlotConfiguration`** A3h branch.

Tag	ASN.1 name	Function	Len	Access
80h	**`exchangeLevel`**	Exchange Level: 01h – TPDU 02h – APDU 04h – Extended APDU	1	RW
Set APDU: FF 70 07 6B 0B **A2 09 A1 07 A3 05 A0 03 80 01 xx** 00 **Response:** 9D 00 90 00 Get APDU: FF 70 07 6B 0A **A2 08 A0 06 A3 04 A0 02 80 00** 00 **Response:** BD 03 80 01 xx 90 00				

Tag	ASN.1 name	Function	Len	Access
82h	**voltageSequence**	Set voltage sequence.	1	RW
Set APDU: FF 70 07 6B 0B **A2 09 A1 07 A3 05 A0 03 82 01 xx** 00 **Response:** 9D 00 90 00 Get APDU: FF 70 07 6B 0A **A2 08 A0 06 A3 04 A0 02 82 00** 00 **Response:** BD 03 82 01 xx 90 00				

Sequence encoding byte is described in chapter 6.2 Voltage Selection.

Tag	ASN.1 name	Function	Len	Access
83h	**operatingMode**	Operating Mode: 00h – ISO/IEC 7816 mode 01h – EMVCo mode	1	RW
Set APDU: FF 70 07 6B 0B **A2 09 A1 07 A3 05 A0 03 83 01 xx** 00 **Response:** 9D 00 90 00 Get APDU: FF 70 07 6B 0A **A2 08 A0 06 A3 04 A0 02 83 00** 00 **Response:** BD 03 80 01 xx 90 00				

Sequence encoding byte is described in chapter 6.3 Data Exchange Level.

11.3 User EEPROM area

KC 1000 SC offers 1024 bytes of EEPROM available for user. This space may be used to store customer configuration data.

There are 3 tags to use when accessing user EEPROM in **readerEEPROM** A7h branch:

Tag	ASN.1 name	Function	Len	Access
81h	**eepromOffset**	Offset on read or write operation	2	RW
82h	**eepromRdLength**	Number of bytes to read from offset	1	R
83h	**eepromWrData**	Data to write on offset in EEPROM	n	W

When specifying command to read or write data **eepromOffset** tag must be specified.

Command to store data in user EEPROM area.

11.3.1 Write

Sample command to write 5 bytes at offset 0:

Tags	Function
FF 70 07 6B 11	vendor command
A2 0F	reader information API
A1 0D	Set
A7 0B	readerEEPROM
81 02 00 00	offset 0000
83 05 01 02 03 04 05 00	5 bytes of data
Write: FF 70 07 6B 11 A2 0F A1 0D A7 0B 81 02 00 00 83 05 01 02 03 04 05 00 **Response**: 9D 00 90 00	

11.3.2 Read

Sample command to read 5 bytes of data from offset 0.

Tags	Function
FF 70 07 6B 11	vendor command
A2 0B	reader information API
A0 09	Get
A7 07	readerEEPROM
81 02 00 00	offset 0000
82 01 05 00	number of bytes to read
Write: FF 70 07 6B 0D A2 0B A0 09 A7 07 81 02 00 00 82 01 05 00 **Response**: 9D 05 01 02 03 04 05 90 00	

11.4 Reader Configuration Control

The **`readerConfigurationControl`** tag A9h is constructed, and the SET branches control the reader's behavior.

Tag	ASN.1 name	Function	Len	Access
80h	**`rebootDevice`**	Reboot device	1	command
Set APDU: FF 70 07 6B 09 **A2 07 A1 05 A9 03 80 01 00** 00 **Response:** 9D 00 90 00				

Appendix: A Enabling Escape CCID commands

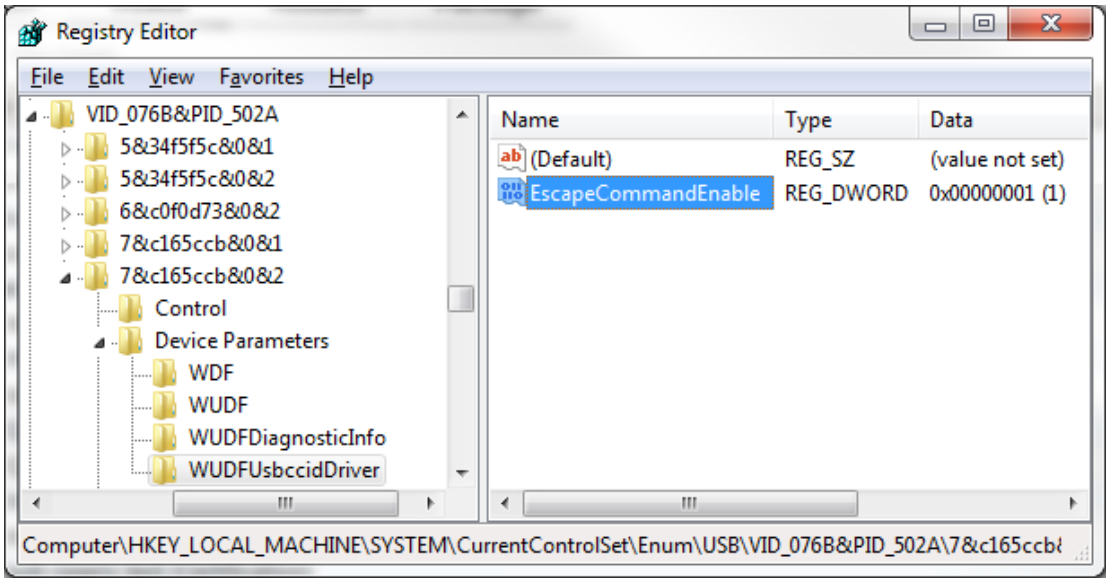
In order to send or receive an Escape command to a reader using Microsoft's CCID driver, add the DWORD registry value **EscapeCommandEnable** and set to a non-zero value under one of the following keys:

- **Windows 7 and 8**
  HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Enum\USB\VID_076B&PID_502A\<serial number>\Device Parameters\WUDFUsbccidDriver
- **Prior Windows 7**
  HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Enum\USB\VID_076B&PID_502A\<serial number>\ Device Parameters

Then the vendor IOCTL for the Escape command is defined as follows:

```
#define IOCTL_CCID_ESCAPE SCARD_CTL_CODE(3500).
```

For details see http://msdn.microsoft.com/en-us/windows/hardware/gg487509.aspx.



If reader with different serial number is connected to computer the operation must be repeated.

Appendix: B Definitions, Abbrev and Symbols

AES	Advanced Encryption Standard
APDU	Application Protocol Data Unit
API	Application Programming Interface
ASN.1	Abstract Syntax Notation One
BER	Basic Encoding Rules
CLA	Class byte of an APDU
DER	Distinguished Encoding Rules
MAC	Message Authentication Code
MSDN	Microsoft® Developer Network
OID	Object Identifier
PAC	Physical Access Control
PACS	PAC Physical Access Control Services
PDU	Protocol Data Unit
PC/SC	Personal Computer/Smart Card
SIO	Secure Identity Object
CSN	Card Serial Number
IFD	Interface Device (for accessing ICC card)

Appendix: C References

[ISO 7816-3]	ISO 7816-3 Identification cards - Integrated circuit cards - Part 3: Cards with contacts - Electrical interface and transmission protocols Third edition - 2006-11-01
[ISO 7816-4]	ISO 7816-4 Identification cards - Integrated circuit cards - Part 4: Organization, security and commands for Interchange Second edition - 2005-01-15
[ISO 8825]	ISO/IEC8825 ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER) Fourth edition 2008-12-15 or X.690 Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)
[PCSC-3-Sup-CL]	Interoperability Specification for ICCs and Personal Computer Systems Part 3. Supplemental Document for Contactless ICCs Revision 2.02.00
[PCSC-3]	Interoperability Specification for ICCs and Personal Computer Systems Part 3. Requirements for PC-Connected Interface Devices Revision 2.01.09
[PCSC-3-Sup]	Interoperability Specification for ICCs and Personal Computer Systems Part 3. Supplemental Document Revision 2.01.09
[PCSC-3-AMD]	Interoperability Specification for ICCs and Personal Computer Systems Part 3. Requirements for PC-Connected Interface Devices - AMENDMENT 1 Revision 2.01.09
[PCSC-10-SPE]	Interoperability Specification for ICCs and Personal Computer Systems Part 10. IFDs with Secure Pin Entry Capabilities Revision 2.02.08
[CCID]	Specification for Integrated Circuit(s) Cards Interface Devices Revision 1.1 April 22rd, 2005

Appendix: D Sample Code

This sample displays card ATR and name of product.

```
// Sample C code that displays ATR and product name.
// Link with winscard.lib
// Copyright 2014, HID Global/ASSA ABLOY AB

#include <stdio.h>
#include <windows.h>
#include <winscard.h>

int main(int argc, char* argv[])
{
    LONG lResult;
    SCARDCONTEXT hContext;
    SCARDHANDLE hCard;
    DWORD dwActiveProtocol;
    DWORD dwLen;
    SCARD_IO_REQUEST pIoReq;
    BYTE pBuffer[32];
    USHORT SW;
    DWORD i;

    //APDU to get data from card
    BYTE PRODUCT_NAME[] = {0xFF, 0x70, 0x07, 0x6B, 0x08, 0xA2, 0x06, 0xA0, 0x04, 0xA0,
                           0x02, 0x82, 0x00, 0x00};

    // First Cherry KC 1000 SC reader name
    WCHAR szReader[] = L"Cherry KC 1000 SC 0";

    // Establish context
    lResult = SCardEstablishContext(SCARD_SCOPE_USER, NULL, NULL, &hContext);
    if( SCARD_S_SUCCESS != lResult )
    {
        printf("SCardEstablishContext failed. Error code 0x%08X.\n", lResult );
        return 1;
    }

    //Connect to card
    lResult = SCardConnect( hContext,
                            szReader,

                            SCARD_SHARE_SHARED,
                            SCARD_PROTOCOL_T0 | SCARD_PROTOCOL_T1,
                            &hCard,
                            &dwActiveProtocol);

    if( SCARD_S_SUCCESS != lResult )
    {
        //release context
        SCardReleaseContext(hContext);

        printf("Can not detect card. Error code 0x%08X.\n", lResult );
        return 1;
    }
```

```
//Select protocol T=1 or T=0
if( SCARD_PROTOCOL_T1 == dwActiveProtocol )
{
    pIoReq = *SCARD_PCI_T1;
}
else
{
    pIoReq = *SCARD_PCI_T0;
}

//get ATR
dwLen = sizeof(pBuffer);
lResult = SCardStatus(  hCard,
                        NULL,
                        NULL,
                        NULL,
                        NULL,
                        pBuffer,
                        &dwLen);

if( SCARD_S_SUCCESS != lResult )
{
    //disconnect card
    SCardDisconnect(hCard, SCARD_LEAVE_CARD);

    //release context
    SCardReleaseContext(hContext);

    printf("Can not get card status. Error code 0x%08X.\n", lResult );
    return 1;
}

//display ATR
printf("ATR: ");
for(i=0;i<dwLen-2;i++)
{
    printf(" %02X", pBuffer[i]); //print hex digits
}
printf("\n"); //end of line

dwLen = sizeof(pBuffer);
lResult = SCardTransmit(hCard,
                        &pIoReq,
                        PRODUCT_NAME,
                        sizeof(PRODUCT_NAME),
                        NULL,
                        pBuffer,
                        &dwLen);

if( SCARD_S_SUCCESS != lResult )
{
    //release context
    SCardReleaseContext(hContext);

    printf("Card not detected. Error code 0x%08X.\n", lResult );
    return 1;
```

```
}

SW = pBuffer[dwLen-2] << 8 | pBuffer[dwLen-1];

printf("SW1SW2: 0x%04X\n", SW );

   //response code
if( SW != 0x9000 )
{
          //disconnect card
          SCardDisconnect(hCard, SCARD_LEAVE_CARD);

          //release context
       SCardReleaseContext(hContext);

    printf("Command not accepted. Error code 0x%04X.\n", SW );
    return 1;
}

//display response
printf("Data:");
for(i=0;i<dwLen-2;i++)
{
    printf(" %02X", pBuffer[i]); //print hex digits
}
printf("\n"); //end of line

printf("ASCII:\n");
for(i=4;i<dwLen-2;i++)
{
    printf(" %02X: %c\n", pBuffer[i], pBuffer[i]); //print hex digits
}
printf("\n"); //end of line

//disconnect card
SCardDisconnect(hCard, SCARD_LEAVE_CARD);

//release context
SCardReleaseContext(hContext);
return 0;
```