Reference Manual for Hirsch R-Series Smart Card Reader Writer Products

SCR3310 V2

SCR3500 Family

uTrust 2700 R

uTrust 2500 R & R EE

Abstract

This document contains in-depth information about the software features of the uTrust R based smart card reader / writer products.

Audience

This document is intended for system integrators and software developers.

Revision History

Rev.	Date	Description
1.0	2022-05-19	First published external version

Information

For additional information, please visit http://www.hirschsecure.com/

Table of Contents

Reference Manual

USB-CCID Smart Card Reader/Writers
R-Series Products

Legal information

Disclaimers

The content published in this document is believed to be accurate. However, Hirsch does not provide any representation or warranty regarding the accuracy or completeness of its content, or regarding the consequences of your use of the information contained herein.

Hirsch reserves the right to change the content of this document without prior notice. The content of this document supersedes the content of any previous versions of the same document. This document may contain application descriptions and/or source code examples, which are for illustrative purposes only. Hirsch gives no representation or warranty that such descriptions or examples are suitable for the application that you may want to use them for.

Should you notice any problems with this document, please provide your feedback to support@hirschsecure.com.

Licenses

If the document contains source code examples, they are provided for illustrative purposes only and subject to the following restrictions:

- You MAY at your own risk use or modify the source code provided in the document in applications you may develop. You MAY distribute those applications ONLY in the form of compiled applications.
- You MAY NOT copy or distribute parts of or the entire source code without prior written consent from Hirsch.
- You MAY NOT combine or distribute the source code provided with Open Source Software or with software developed using Open Source Software in a manner that subjects the source code or any portion thereof to any license obligations of such Open Source Software.

If the document contains technical drawings related to Hirsch products, they are provided for documentation purposes only. Hirsch does not grant you any license to its designs.

Trademarks, logos and brand names

All trademarks, logos, and brand names are the property of their respective owners.

Concerned R-Series Products

- SCR3310 V2

- uTrust 2500 R
- uTrust 2500 R EE

- uTrust 2700 R

- SCR3500 Family
  USB-A, USB-B, and USB-C

Objective of the manual and target audience

The manual targets solution providers. It assumes knowledge about ISO/IEC 7816 and commonly used engineering terms.

This manual provides details of the Firmware features of the SCR3310V2, uTrust 2700 R and SCR3500 Smartfold family smart card reader/writer products and targets application developers.

Definition of various terms and acronyms

Term	Expansion
APDU	Application Protocol Data Unit
ATR	Answer to Reset, defined in ISO/IEC 7816
Byte	Group of 8 bits
CCID	Chip Card Interface Device
CID	Card Identifier
ESD	Electrostatic Discharge
LED	Light emitting diode
NA	Not applicable
NAD	Node Address
Nibble	Group of 4 bits. 1 digit of the hexadecimal representation of a byte. *Example:* 0xA3 is represented in binary as (10100011)b. The least significant nibble is 0x3 or (0011)b and the most significant nibble is 0xA or (1010)b
PC/SC	Personal Computer/Smart Card: is an interoperability standard ensuring the communication between computers and smartcards
PID	Product ID - is a unique number that helps identify a hardware product
RFU	Reserved for future use
USB	Universal Serial Bus
VID	Vendor ID - is a unique 32-bit number identifying the manufacturer of a product
(xyz)b	Binary notation of a number x, y, z $\in$ {0,1}

0xYY	The byte value YY is represented in hexadecimal

References

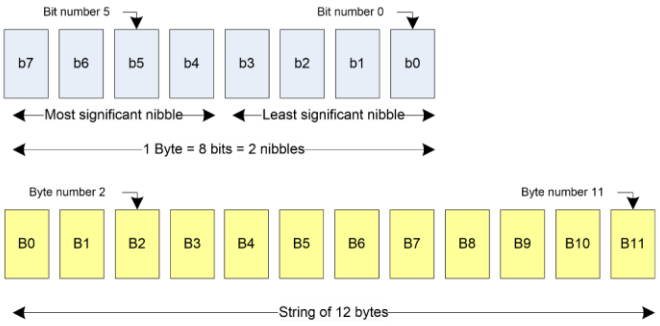
Doc ref in the manual	Description	Issuer
ISO/IEC 7816-3	Identification cards - Integrated circuit cards - Part 3: Cards with contacts — Electrical interface and transmission protocols	ISO / IEC
ISO/IEC 7816-4	Identification cards - Integrated circuit cards - Part 4: Organization, security and commands for interchange	ISO / IEC
PC/SC	Interoperability Specification for ICCs and Personal Computer Systems v2.01.14	PC/SC Workgroup
CCID	Specification for Integrated Circuit(s) Cards Interface Devices 1.1	USB-IF
USB	Universal Serial Bus Specification 2.0	USB-IF

Conventions for bits and bytes

Bits are represented by a lower case '**b**' followed by an ordering digit which indicates its position.

Bytes are represented by an upper case '**B**' followed by one or more ordering digits which indicate its position.

Bit and Byte representation



Example: 163 decimal number representation

DECIMAL	HEXADECIMAL	BINARY
163	0xA3	10100011
least significant nibble	0x3	0011
most significant nibble	0xA	1010

General information about R type products

Common Key features

- ISO/IEC 7816 compliant smart card reader/writer
- PC/SC v2.0 compliant
- 249 bytes of non-volatile user memory

Applications

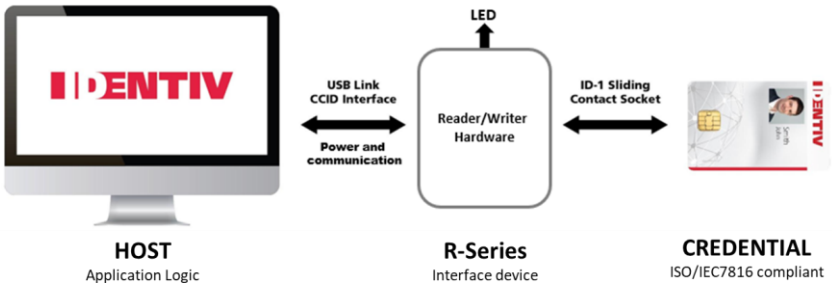
General

Hirsch smart card reader/writer products interface a personal computer host application supporting PC/SC interface with smart cards according to ISO/IEC 7816 as well as with synchronous memory cards like CAC and PKI cards or banking cards and health insurance cards. The reader firmware is handling the communication protocol but not the application related to the credential. The application-specific logic has to be implemented by software developers on the host.

Applications provided by Hirsch

Hirsch does not provide PKI or CAC applications. Hirsch only provides a few applications for development, test and evaluation purposes that function with its smart card reader/writers. Developer tools can be found on the Hirsch support page.

R-Series High Level Architecture

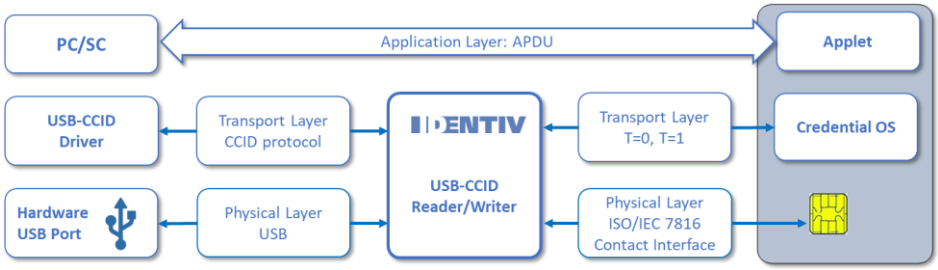


The R-Series devices offer the following interfaces.

- LED user interface for reader/writer status indication
- Contact smart card interface
- USB-CCID host interface The link between R series products and the host is the USB-CCID interface providing both the power and the communication channel. The device Microcontroller contains the firmware developed by Hirsch and handles the communication between the host application and the inserted credential as well as the user interface.

Software architecture

The Hirsch R-Series smart card reader/writers leverage a PC/SC CCID driver which is available for Windows, macOS X and Linux operating systems.
Applications can interface with the driver directly through the PC/SC interface.



With the diverse distributed Linux derivatives, there may be distribution specific drivers that should get installed using the install mechanism of the used Linux distribution.

If there is no driver available, a CCID Linux driver may be downloaded from the webpage of the maintainer, Ludovic Rousseau, https://ccid.apdu.fr/ or here at Debian .
Additionally, Hirsch provides a proprietary driver for all the supported OSs.

Hirsch drivers can be downloaded from the product support page:
https://support.hirschsecure.com/

LED behavior

Depends on the chosen product of the R-Series. Either a single or a bi-color LEDs is available.

SCR3310 V2, SCR3500 SmartFold	LED Single color
Reader powered, card out	*OFF*
Reader powered, card in but not powered	*OFF*
Card powered	*ON*
Card access	*Blinking: 500ms ON / 500ms OFF*
Error condition	*Blinking: 100ms ON / 100ms OFF*
uTrust 2500 R/R EE , uTrust 2700 R	**LED Bi-color**
Reader powered, no card inserted	*OFF*
Reader powered, card inserted, not powered	*OFF*
Card powered	*ON*
Card access	*LED1 yellow/green* *Blinking: 500ms ON / 500ms OFF*
Error condition	*LED2 red* *Blinking: 100ms ON / 100ms OFF*

USB related information

Parameter	Description / Value	
USB	*Bus powered*	
USB specification	*USB 2.0 Full Speed*	
USB Speed	*Full Speed Device (12Mbit/s)*	
USB Device Class = CCID	*PID*	*VID*
SCR3310 V2	*0x5116*	*0x04E6*
uTrust 2500 R xx	*0x5710*	*0x04E6*
uTrust 2700 R	*0x5810*	*0x04E6*
SCR3500 USB-C	*0x581D*	*0x04E6*
SCR3500 USB-A	*0x581C*	*0x04E6*

Smart Card interface

Parameter	Description / Value
Smart card operating frequency	*up to 12MHz*
Maximum supported card baud-rate	*600Kbps*
Cards supported	*Class A, B and C asynchronous smart cards with T=0 or T=1 protocol* *Synchronous smart cards (2wire, 3wire, I²C)*
ISO/IEC 7816 compliant	*Yes*
EMV 4.2 compliant	*Yes*
CT-API compliant	*Yes*
Number of slots	*Single ID-1 smart card slot*
Ejection mechanism	*Manual*

Software modules

Installation

On Operating Systems with a PC/SC USB-CCID driver preinstalled, no installation is necessary.

Where there's no PC/SC CCID driver preinstalled (Linux systems) the driver has to be installed using a distribution specific measures or installed using the available source packages.

Utilities and Diagnostic Tools for Smart Card Readers

Description	Operating System	
Smart PC/SC Diagnostic This utility enables to check card reader configuration and create a log file.	Windows All	Download
Fix PC/SC Resource Manager This tool repairs a damaged PC/SC Resource Manager.	Up to Windows 7	Download
TestResMan This utility enables testing the PC/SC Resource Manager in Windows.	Windows All	Download
TestResMan This utility enables testing the PC/SC API (pcsc-lite) in Linux.	Linux	Download
CT-API test utility This utility provides a test scenario for the CT-API for Linux	Linux	Download
CT-API test utility This utility provides a test scenario for the CT-API for Mac OS X	Mac OS X	Download

Commented [1]: Need to be moved to Hirsch database and links updated.

Commented [2]: Need to be moved to Hirsch database and links updated.

Commented [3]: Need to be moved to Hirsch database and links updated.

Commented [4]: Need to be moved to Hirsch database and links updated.

Commented [5]: Need to be moved to Hirsch database and links updated.

Commented [6]: Need to be moved to Hirsch database and links updated.

USB CCID Driver

USB device listing

PC/SC applications name can be one of the following depending on the reader model. However, the names could change depending on the driver used (on Windows) or the name listed against the device in the Info.plist file. The recommended way for applications and middleware to work consistently is to issue a SCardListReaders() call and pull the IFD name from the result.

Identiv uTrust 2500 (R or R-EE) Smart Card Reader
Identiv uTrust 2700 R Smart Card Reader
Identive uTrust 2700 R Smart Card Reader
SCM Microsystems SCR3310 Smart Card Reader

Supported operating systems

See previous chapter and consult website and latest data sheets for up to date information.

CT-API

A CT-API interface that mostly is used in German banking applications and in conjunction with health insurance cards, is available for the reader.

MCard-API

With this proprietary Hirsch API, it is possible to access a vast majority of synchronous memory cards.

Supported memory cards					
SLE4404	*SLE4428*	*SLE4432*	*SLE4436*	*SLE6636*	*SLE4442*
SLE5532	*SLE5536*	*SLE5542*	*AT24C01ASC*	*AT24C02SC*	*AT24C04SC*
AT24C08SC	*AT24C16SC*	*AT24C32SC*	*AT24C64SC*	*AT24C128SC*	*AT24C256SC*
AT24C512SC	*AT88SC153*	*AT88SC1608*	*ST14C02*		

Firmware

CCID transport protocol

The R-series product firmware implements a transport protocol that is compliant with USB Device Class: *Smart Card CCID Specification for Integrated Circuit(s) Cards Interface Devices Revision 1.10*. This paragraph describes the CCID specification features that are implemented.

Supported CCID class requests

Abort

Supported CCID messages

The following CCID messages are supported for the contact interface when received through bulk-out endpoint.

PC_to_RDR_IccPowerOn	*PC_to_RDR_IccPowerOff*	*PC_to_RDR_GetSlotStatus*
PC_to_RDR_XfrBlock	*PC_to_RDR_GetParameters*	*PC_to_RDR_SetParameters*
PC_to_RDR_Escape	*PC_to_RDR_Abort*	*PC_to_RDR_NotifySlotChange*
PC_to_RDR_ResetParameters	*PC_to_RDR_T0APDU*	*PC_to_RDR_SetDatarateAndClockFrequency*

CCID Error Codes

Extensive error codes are reported on many conditions during all CCID responses.
Most of the error messages are reported by the CCID appropriately.
Some of the main error codes for the contact interface are listed below.

HW_ERROR	*XFR_PARITY_ERROR*	*ICC_PROTOCOL_NOT_SUPPORTED*
BAD_ATR_TS	*BAD_ATR_TCK*	*ICC_MUTE*
CMD_ABORTED	*Command not supported*	

The following subsections discuss when and why these error codes are returned:

HW_ERROR	This error code is returned when a hardware short circuit condition is detected, during application of power to the card or if any other internal hardware error is detected.
XFR_PARITY_ERROR	This error code is returned when a parity error condition is detected. This error will be reported in the response to a PC_to_RDR_XfrBlock message.
ICC_PROTOCOL_NOT_SUPPORTED	This error code is returned if the card is signaling to use a protocol other than T=0 or T=1 in its ATR.
BAD_ATR_TS	This error code is returned if the initial character of the ATR contains invalid data.
BAD_ATR_TCK	This error code is returned if the check character of the ATR contains is invalid.
ICC_MUTE	This error code is returned when the card does not respond until the reader timeout occurs. This error will be reported in the response to PC_to_RDR_XfrBlock message and PC_to_RDR_IccPowerOn messages.
CMD_ABORTED	This error code is returned if the command issued has been aborted by the control pipe.
Command not supported	This error would be returned, if the command would not be supported by the reader.

Commands description

Escape commands for the uTrust R-Series

Sending Escape commands to uTrust R

A developer can use the following method to send Escape commands to uTrust R products

- SCardControl method defined in PC/SC API

In Windows, in order to be able to send Escape commands to the uTrust R product series, the feature has to be enabled by setting a REG_DWORD value named 'EscapeCommandEnable' in the registry to a value of '1'.

For Windows XP and Windows Vista, the key to hold the value would be HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Enum\USB\VID_04E6&PID_5810\ Device-Instance-xxxx \Device Parameters

For Windows 7, Windows 8.1, Windows 10, Win11 KEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Enum\USB\VID_04E6&PID_5810\Device-Instance-xxxx \Device Parameters\WUDFUsbccidDriver

Device-Instance-xxxx has got to be equal to the serial number of the reader used, so this modification has got to be made for every physical reader intended to be used on the machine in question. The reader has got to be plugged in at least once for the mentioned key to exist and the driver has got to be restarted for this setting to take effect. (Unplug and replug the reader).

To be able to work with synchronous memory cards using our MCard API, the same setting will have to be established.

See appendix B for some sample code sending Escape commands to the reader.

Escape command codes

Escape commands can be used by an application to configure uTrust xxxx R to function in a mode that is not its default configured mode or to get specific information. To put the uTrust R product back into its default mode, it either has to be unplugged and plugged again or the application can send the same Escape command again.

The following Escape commands are supported by uTrust R series products:

Escape command	Code
READER_SETMODE	*0x01*
READER_GETMODE	*0x02*
CONTACT_GET_SET_POWERUPSEQUENCE	*0x04*
CONTACT_EMV_LOOPBACK	*0x05*
CONTACT_EMV_SINGLEMODE	*0x06*
CONTACT_APDU_TRANSFER	*0x08*
CONTACT_CONTROL_PPS	*0x0F*
CONTACT_EXCHANGE_RAW	*0x10*
READER_GETIFDTYPE	*0x12*
READER_LED_CONTROL	*0x19*
READER_LED_CONTROL_BY_FW	*0xB2*
READER_GETINFO_EXTENDED	*0X1E*
CONTACT_GET_SET_CLK_FREQUENCY	*0x1F*
CONTACT_GET_SET_ETU	*0x80*
CONTACT_GET_SET_WAITTIME	*0x81*
CONTACT_GET_SET_GUARDTIME	*0x82*
CONTACT_GET_SET_MCARD_TIMEOUT	*0x85*
CONTACT_CONTROL_ATR_VALIDATION	*0x88*

READER_SETMODE

This Escape command sets the current mode of the reader. Applications may call this function, to set the desired mode. Typically, this call is used to switch between the ISO/IEC 7816, EMV and memory card operations. Upon power on, the reader will reset to the default ISO/IEC 7816 mode.

Input: The first byte of the input buffer contains the Escape code value and the second one will contain the value for the desired mode of operation. The output buffer field will be NULL.

Byte0	Byte1
Escape code (0x01)	*Mode*

The following table gives the value of modes as interpreted by the firmware:

Mode	Value	Remarks
ISO	*0x00*	*ISO/IEC 7816 mode*
EMV	*0x01*	*EMV*
Synchronous	*0x02*	*memory card mode (Synchronous)*

ISO mode uses APDU mode of data transfer and is used for normal operations. This is the default mode of the reader upon power up.

EMV mode also uses APDU mode of data transfer and is used for EMV test purposes. This mode has more stringent checks for smart card detection and communication as per EMV4.2 spec.

Synchronous mode is used for communicating only with memory cards. Any other value sent as mode is invalid.
Output buffer

Output buffer
NULL

READER_GETMODE

This Escape command may be used to retrieve the current mode of the reader.

The input buffer is

Byte0
Escape code(0x02)

Output: Current active reader mode will be returned as a BYTE value as is interpreted by reader firmware as follows

Mode	Value	Remarks
ISO	*0x00*	*ISO/IEC 7816 mode*
EMV	*0x01*	*EMV*
Synchronous	*0x02*	*memory card mode (synchronous)*

CONTACT_GET_SET_POWER_UP_SEQUENCE

This Escape command is used by the application/driver to get/set the following parameters:

- Smart card Power-on sequence
- Delay between successive Activation retries
- Enable/Disable any Voltage Class

As soon as card insertion is detected and power on message is received from the host, the firmware will start activation with the configured voltage sequence. If the activation fails, it will wait for the configured activation delay and then retry with the next enabled voltage class. If power up succeeds at an operating voltage, the firmware will continue card communication at that voltage. If power up fails in all the enabled operating voltages, then the firmware will report an error. The default power-up sequence would be A – B – C.

Input: The first byte of the input buffer contains the Escape code. The next byte contains the function to be performed. The third byte contains the parameter for the function.

Byte0	Byte1		Byte2
	Value	Description	
Escape code (0x04)	*0x00*	*Starts with Class C voltage. (1.8V – 3V – 5V order)*	-
	0x01	*Starts with Class A voltage. (5V – 3V – 1.8V order)*	-
	0x08	*Time delay between resets*	*Delay value in milliseconds*
	0x09	*Enable/Disable a Voltage Class*	*Bit Map of all Voltage Classes [Bit0 – Class A; Bit1 – Class B; Bit2 – Class C] Set bit to enable the Voltage class Clear bit to disable the Voltage class*
	0xFE	*Retrieves all the above values*	-
	0xFF	*Retrieves the current Power up sequence*	-

For retrieving all settings (0xFE), the output will be the following:

Byte0		Byte 1	Byte2
Value	Description		
0x00	*Starts with Class C voltage. (1.8V – 3V – 5V order)*	*Time delay between resets in milliseconds*	*Bit Map of all Voltage Classes [Bit0 – Class A; Bit1 – Class B; Bit2 – Class C]*
0x01	*Starts with Class A voltage. (5V – 3V – 1.8V order)*		

For retrieving current power up sequence (0xFF), the output will be:

Byte0	
Value	Description
0x00	*Starts with Class C voltage. (1.8V – 3V – 5V order)*
0x01	*Starts with Class A voltage. (5V – 3V – 1.8V order)*

Example: retrieve all the current settings:
DataIn = **04 FE**
DataOut: **01 0A 07** (3 bytes)

00: Starting with Class A
0A: 10ms delay between resets
07: Class A, B, and C enabled

CONTACT_EMV_LOOPBACK

This Escape command lets the host force the firmware to perform an EMV Loop-back application.

The input buffer is

Byte0
Escape code(0x05)
Output buffer
NULL

CONTACT_EMV_SINGLEMODE

This Escape command lets the host perform a one-shot EMV Loop-back application as specified in the EMV Level 1 Testing Requirements document.

Input:

Byte0
Escape code(0x06)
Output buffer
NULL

CONTACT_APDU_TRANSFER

This Escape command exchanges a short APDU with the smart card. The user has to ensure that a card is inserted and powered before issuing this Escape command.
This Escape command mostly is used by the MCard API to access synchronous memory cards.

Input: The input buffer contains the Escape code value followed by the short APDU to be sent to the card.

Byte0	Byte1 onwards
Escape code(0x08)	*Short APDU to be sent to card*
Output buffer	
Response APDU	

CONTACT_CONTROL_PPS

This Escape command enables or disables the PPS done by the firmware/device for smart cards. This setting will take effect from the next card connect and remains effective till it is changed again or the next Reader power on. Default mode is PPS enabled.

Input: The first byte of input buffer contains the Escape code and the following byte, if 1 disables the PPS and if 0 enables the PPS.

Byte0	Byte1
Escape code(0x0F)	*PPS control byte (1-DISABLES PPS, 0-ENABLES PPS)*
Output buffer	
NULL	

CONTACT_EXCHANGE_RAW

This Escape command can be used to perform raw exchange of data with the card. The user must ensure that a card is inserted and powered on before issuing this Escape command. The Card is deactivated upon any reception error.

Input: The input buffer for this command will contain the Escape code, low byte of the length of data to be sent, high byte of length of data to be sent, low byte of the length of expected data, high byte of length of expected data and the command.

Byte0	Byte1	Byte2	Byte3	Byte4	Byte 5 onwards
Escape code(0x10)	*LSB of send length*	*MSB of send length*	*LSB of expected length*	*MSB of expected length*	*Raw data to the card*
Output buffer					
Response APDU					

READER_GET_IFDTYPE

This Escape command is used to get the current IFD type from the reader.

Input: The first byte of the input buffer contains the Escape code.

Byte0
Escape code(0x12)

Output: The reader returns the PID of the firmware which can be used to identify the reader.

PID value		Description
B0	B1	
0x10	*0x58*	*USB PID of Hirsch uTrust 2700 R smart card Reader*

READER_LED_CONTROL

This Escape command may be used to toggle the LED state. LED control by firmware should be disabled using the Escape command READER_LED_CONTROL_BY_FW to see proper LED change while using this IOCTL else the LED state will be overwritten by the FW LED behavior.
Input: The first byte of the input buffer contains the Escape code, followed by the LED number always set to 0 (just one LED) and then the desired LED state.

Byte0	Byte 1	Byte2
Escape code(0x19)	*LED number (0 GREEN)*	*LED state (0-OFF, 1-ON)*
Output buffer		
NULL		

READER_LED_CONTROL_BY_FW

This command is used to enable/disable LED control by firmware. Default setting is: LED is controlled by firmware.
Input: The first byte of the input buffer contains the Escape code. The second byte specifies if LED control by the firmware should be disabled or enabled. Output buffer NULL.

Byte0	Byte1	
	Value	Description
Escape code(0xB2)	*0*	*Enable LED Control by firmware*
	1	*Disable LED Control by firmware*
Get State	*FF*	*0 -- LED control by firmware enabled* *1 -- LED control by firmware disabled*
Output buffer		
NULL or current state	*No response is returned for set state. For Get State 1 byte response is received.*	

READER_GET_INFO_EXTENDED

This Escape command may be used to retrieve extended information about the reader and supported features.

Input: The first byte of the input buffer contains the Escape code.

Byte0
Escape code(0x1E)

The firmware returns data as per structure SCARD_READER_GETINFO_PARAMS_EX mentioned below. This Escape command is used to get the firmware version, reader capabilities, and Unicode serial number of the reader.

Field Size [Bytes]	Field Name	Field Description	Default value
1	*byMajorVersion*	*Major Version in BCD*	*Based on current firmware version*
1	*byMinorVersion*	*Minor Version in BCD*	
1	*bySupportedModes*	*Total no of supported modes in the reader*	*0x03 (ISO, EMV and MCard modes)*
2	*wSupportedProtocols*	*Protocols supported by the Reader* *Bit 0 – T0* *Bit 1 – T1*	*0x0300 (LSB first)*
2	*winputDevice*	*IO_DEV_NONE 0x00* *IO_DEV_KEYPAD 0x01* *IO_DEV_BIOMETRIC 0x02*	*0x0000(LSB first)*
1	*byPersonality*	*Reader Personality (Not Used)*	*0x00*
1	*byMaxSlots*	*Maximum number of slots*	*0x01 (Single slot device)*
1	*bySerialNoLength*	*Serial number length*	*0x1C*
28	*bySerialNumber [28]*	*Unicode serial number*	*Reader serial number(MSB first)*

DataIn = **1E**

DataOut: **01 00 03 03 00 00 00 00 01 1C 35 00 33 00 36 00 39 00 31 00 33 00 30 00 31 00 32 00 30 00 30 00 30 00 36 00 32 00** (38 bytes)

CONTACT_GET_SET_CLK_FREQUENCY

In case when an application wants to get or set the smart card clock frequency, this Escape command is used to instruct the reader to change the clock or to get the current Clock divisor used. Once set, the change in frequency will take effect immediately. Default divisor value is 10, that is 4.8MHz.

Input: The first byte of the input buffer will contain the Escape code; the next byte will contain the clock divisor value to set clock frequency or 0xFF to get clock frequency.

Byte0	Byte1	
	Value	Description
Escape code(0x1F)	*Clock divisor*	*The value to be set in the smart card CLK divisor register*
	0xFF	*Get current Clock divisor value*

Set clock frequency: None

Get clock frequency: One byte value indicating the current Clock divisor.

Output buffer
NULL or current divisor

Clock Divisor values:

Value	Frequency
12	4 MHz
10	4.8 MHz
8	6 MHz
7	6.8 MHz
6	8 MHz
5	9.6 MHz
4	12 MHz
3	16 MHz

DataIn = **1F FF**
DataOut: **0A** (1 byte)

CONTACT_GET_SET_ETU

This Escape command is used by the HOST to get/set the current ETU for smart cards. Once set, the new ETU value will take effect immediately.

Input: The input buffer contains the Escape followed by an 8 bit GET/SET identifier. For SET ETU, a DWORD specifying the value to be set is as follows.

Byte0	Byte1		Byte2	Byte3	Byte4	Byte5
	Value	Description	Wait time			
Escape code(0x80)	*0x01*	*SET ETU*	*BIT31-BIT24*	*BIT23-BIT16*	*BIT15-BIT8*	*BIT7-BIT0*
	0x00	*GET ETU*	-	-	-	-

For both Set and Get ETU, the output will be the following.

Byte0	Byte1	Byte2	Byte3
ETU value			
BIT31-BIT24	*BIT23-BIT16*	*BIT15-BIT8*	*BIT7-BIT0*

DataIn = **80 00**
DataOut: **00 00 01 40** (4 bytes)

CONTACT_GET_SET_WAITTIME

This Escape command is used to get/set the Character/Block Waiting Time for smart cards. The wait time is specified in terms of ETU. Once set, the new Wait time will take effect from the next card communication.
Input: The input buffer contains the Escape code followed by an 8 bit GET/SET identifier, an 8 bit Wait time identifier and a 32 bit Wait time value. BWT must be specified in units of 1.25ms and CWT in units of ETU.

Byte0	Byte1		Byte2		Byte3	Byte4	Byte5	Byte6
	Value	Description	Value	Description	Wait time in ETU			
Escape code(0x81)	*0x01*	*SET Wait time*	*0x00*	*CWT*	*BIT31-BIT24*	*BIT23-BIT16*	*BIT15-BIT8*	*BIT7-BIT0*
			0x01	*BWT*				
	0x00	*GET Wait time*	*0x00*	*CWT*	-	-	-	-
			0x01	*BWT*				

For both Get/Set Wait time, the output will be the following.

Byte0	Byte1	Byte2	Byte3
Wait time in ETU			
BIT31-BIT24	*BIT23-BIT16*	*BIT15-BIT8*	*BIT7-BIT0*

DataIn = **81 00 01**
DataOut: **00 00 03 5D** (4 bytes)

CONTACT_GET_SET_GUARDTIME

This Escape command is used to get/set the Character/Block Guard Time of the reader. The guard time is specified in terms of ETU. Once set, the new Guard time will take effect immediately.

Input: The input buffer contains the Escape code followed by an 8 bit GET/SET identifier, an 8 bit guard time identifier and a 32 bit guard time value in ETU.

Byte0	Byte1		Byte2		Byte3	Byte4	Byte5	Byte6
	Value	Description	Value	Description	Guard time in ETU			
Escape code (0x82)	*0x01*	*SET Guard time*	*0x00*	*CGT*	*BIT31-BIT24*	*BIT23-BIT16*	*BIT15-BIT8*	*BIT7-BIT0*
			0x01	*BGT*				
	0x00	*GET Guard time*	*0x00*	*CGT*	*-*	*-*	*-*	*-*
			0x01	*BGT*				

For Get/Set guard time, the output will be the Character/Block Guard Time value.

Byte0	Byte1	Byte2	Byte3
Character Guard time in ETU			
BIT31-BIT24	*BIT23-BIT16*	*BIT15-BIT8*	*BIT7-BIT0*

DataIn = **82 00 01**
DataOut: **00 00 00 18** (4 bytes)

CONTACT_GET_SET_MCARD_TIMEOUT

This Escape command is used to get or set the delay which is applied after a Write operation to memory cards. The delay is specified in milliseconds.
Input: The first byte of the input buffer will contain the Escape code; the next byte will contain the memory card write delay in seconds.

Byte0	Byte1	
	Value	Description
Escape code(0x85)	*0x01*	*Delay in milliseconds for memory card Write*
	Any value other than 1	*Read the current applied delay for memory card Write*

Write delay: No response byte
Read delay value: A byte value specifying the current delay applied during memory card Write in milliseconds

Byte0
Delay in ms

DataIn = **85 00**
DataOut: **00** (1 byte)

CONTACT_CONTROL_ATR_VALIDATION

This Escape command is used to enable or disable the ATR validation by the firmware in ISO/IEC 7816 mode.
In case the card would emit an ATR that is not ISO/IEC 7816 compliant, the card reader may fail to power up the card. In these cases, disabling ATR validation will let you work with the card regardless of ISO conformity of the ATR. By default, ATR validation is enabled.

Input: The first byte of the input buffer will contain the Escape code; the next byte will contain the control byte.

Byte0	Byte1	
	Value	Description
Escape code(0x88)	*0x00*	*Enable ATR validation*
	0x01	*Disable ATR validation*
Output buffer		
NULL		

Annexes

A – Status words table

SW1	SW2	Description
0x90	*0x00*	*NO ERROR*
0x67	*0x00*	*LENGTH INCORRECT*
0x6D	*0x00*	*INVALID INSTRUCTION BYTE*
0x6E	*0x00*	*CLASS NOT SUPPORTED*
0x6F	*0x00*	*UNKNOWN COMMAND*
0x63	*0x00*	*NO INFORMATION GIVEN*
0x65	*0x81*	*MEMORY FAILURE*
0x68	*0x00*	*CLASS BYTE INCORRECT*
0x6A	*0x81*	*FUNCTION NOT SUPPORTED*
0x6B	*0x00*	*WRONG PARAMETER P1-P2*

Annex B

Sample code using Escape commands through Escape IOCTL.

Example for uTrust 2700 R

File Name : uTrust 2700 R Escape.h

```
#ifndef _uTrust_2700 R_ESCAPE_H_
#define _uTrust_2700 R_ESCAPE_H_

#ifdef _cplusplus
extern "C" {
#endif

# pragma pack (1)
        typedef struct {
                BYTE byMajorVersion;
                BYTE byMinorVersion;
                BYTE bySupportedModes;
                WORD wSupportedProtocols;
                WORD winputDevice;
                BYTE byPersonality;
                BYTE byMaxSlots;
                BYTE bySerialNoLength;
                BYTE abySerialNumber[28];
        } ReaderInfoExtended;
# pragma pack ()

#define IOCTL_CCID_ESCAPE                       SCARD_CTL_CODE (0xDAC)
#define READER_SET_MODE                         0x01
#define READER_GET_MODE                         0x02
#define CONTACT_GET_SET_POWERUPSEQUENCE         0x04
#define CONTACT_EMV_LOOPBACK                    0x05
#define CONTACT_EMV_SINGLEMODE                  0x06
#define CONTACT_APDU_TRANSFER                   0x08
#define CONTACT_CONTROL_PPS                     0x0F
#define CONTACT_EXCHANGE_RAW                    0x10
#define READER_GETIFDTYPE                       0x12
#define READER_LED_CONTROL                0x19
#define READER_LED_CONTROL_BY_FW          0xB2
#define READER_GETINFO_EXTENDED                 0x1E
#define CONTACT_GET_SET_CLK_FREQUENCY     0x1F
#define CONTACT_GET_SET_ETU                     0x80
#define CONTACT_GET_SET_WAITTIME          0x81
#define CONTACT_GET_SET_GUARDTIME               0x82
#define CONTACT_GET_SET_MCARD_TIMEOUT     0x85
#define CONTACT_CONTROL_ATR_VALIDATION    0x88

#ifdef __cplusplus
}
#endif

#endif
```

File Name : uTrust 2700 R Escape.c

```
#include <windows.h>
#include <winbase.h>
#include <stdio.h>
#include <conio.h>
#include "winscard.h"
#include "winerror.h"
#include "uTrust 2700R Escape.h"

VOID main(VOID)
{
	SCARDCONTEXT		ContextHandle;
	SCARDHANDLE		CardHandle;
	ReaderInfoExtended	strReaderInfo;
	BYTE			InByte, i;
	DWORD			BytesRead, ActiveProtocol;
	ULONG			ret;
	char			*ReaderName[] = { "Identiv uTrust 2700 R Smart Card Reader 0",NULL };
/*****************************************************************************************************************/
	ContextHandle = -1;
	ret = SCardEstablishContext(SCARD_SCOPE_USER, NULL, NULL, &ContextHandle);
	if (ret == SCARD_S_SUCCESS) {
	ret = SCardConnect(ContextHandle,
		ReaderName[0],
		SCARD_SHARE_DIRECT,
		SCARD_PROTOCOL_UNDEFINED,
		&CardHandle,
		&ActiveProtocol);
		if (ret == SCARD_S_SUCCESS)
		{
		InByte = 0x1E;
		ret = SCardControl(CardHandle,
		IOCTL_CCID_ESCAPE,
		&InByte, 1, &strReaderInfo,sizeof(strReaderInfo), &BytesRead);

		if (SCARD_S_SUCCESS == ret) {
				printf("major version:\t\t%d%d\n", (strReaderInfo.byMajorVersion & 0xF0)
				>> 4, (strReaderInfo.byMajorVersion & 0x0F));
				printf("minor version:\t\t%d%d\n", (strReaderInfo.byMinorVersion & 0xF0)
				>> 4, (strReaderInfo.byMinorVersion & 0x0F));
				printf("modes:\t\t\t%d\n", strReaderInfo.bySupportedModes);
				printf("protocols:\t\t%04x\n", strReaderInfo.wSupportedProtocols);
				printf("input device:\t\t%04x\n", strReaderInfo.winputDevice);
				printf("personality:\t\t%d\n", strReaderInfo.byPersonality);
				printf("max slots:\t\t%d\n", strReaderInfo.byMaxSlots);
				printf("serial no length:\t%d\n", strReaderInfo.bySerialNoLength);
				printf("serial no:\t\t");
				for (i = 0; i < strReaderInfo.bySerialNoLength; i++)
				printf("%c", strReaderInfo.abySerialNumber[i]);
			}
			else { printf("SCardControl failed: %08X\n", ret);
			}
		}
		else { printf("SCardConnect failed: %08X\n", ret);
		}
		ret = SCardReleaseContext(ContextHandle);
	}
	else {
		printf("\n SCardEstablishContext failed with %.8lX", ret);
	}
	printf("\npress any key to close the test tool\n");
getch();
}
```