13.56 MHz RFID USB READER

REFERENCE MANUAL

ubisys®

1. Overview

Welcome to your ubisys RFID reader solution!

This reference manual provides operating and maintenance instructions, command references, a tag compliance matrix, and other detailed product information. If you have any questions or need additional support, drivers, software, libraries or source code samples, please visit our RFID support pages.

http://www.ubisys.de/rfid/support.html

2. Contents

1. Overview 2
2. Contents 3
3. Features 5
4. Supported TAGs 7
5. CDC Firmware 9
   5.1. Features 9
   5.2. CDC/ACM AT Command set 9
      5.2.1. Port Settings 9
      5.2.2. General Frame Format 9
      5.2.3. Device Information 9
      5.2.4. Echo Control 10
      5.2.5. Light-Emitting Diode Control 10
      5.2.6. Buzzer Control 11
      5.2.7. Inventory Scan 11
      5.2.8. Inventory Scan (without Anti-Collision) 12
      5.2.9. Select Tag 12
      5.2.10. Get System Information 13
      5.2.11. Read Data Block 14
      5.2.12. Write Data Block 14
      5.2.13. Lock Data Block 15
      5.2.14. Enable/Disable RF field 17
      5.2.15. Query RF field status 17
      5.2.16. Automatic Scan Mode 18
6. HID Firmware 19
   6.1. Features 19
   6.2. Operation 19
7. CCID Firmware 20
   7.1. Driver Installation 20
   7.2. Features and Tag Support 22
   7.3. ATR Strings 23
      7.3.1. ATR for ISO14443-4 23
      7.3.2. ATR for Memory Cards (ISO15693, ISO14443-3 Type A) 23

7.4. APDU Format 24
7.4.1. Request APDU 24
7.4.2. Response APDU 24
7.5. APDU Definitions 25
7.5.1. Tag Identification Commands 25
7.5.2. Configuration commands 25
7.5.3. Memory Card Access Commands 29
7.5.4. Extended ISO15693 Information Commands 30
7.5.5. ISO15693 Transparent mode 31
8. Firmware update 33
9. ubisys RFID/Smart Card API 40
9.1. Overview 40
9.2. Basic RFID/Smart Card API 40
9.2.1. CSmartCardContext 40
9.2.2. CSmartCard 41
9.2.3. CSmartCardTransaction 42
9.2.4. CSmartCardStatusReceiver 42
9.2.5. CSmartCardStatusListener 43
9.2.6. CSmartCardExceptionHelper 44
9.2.7. Pre-defined Exception Classes 45
10. Physical Dimensions 47
11. Ordering Information 50
11.1. USB Stick 50
11.2. Desktop Pro 50
11.3. Desktop 51
12. Declaration of Conformity 52
13. Document revision history 53
14. Contact 54

3. Features

- Versatile reader supporting three USB modes of operation: CCID = smart card reader, HID = keyboard emulation, CDC/ACM = virtual serial port. Features out-of-the box OS support, i.e. no driver installation on Microsoft Windows, Apple MacOS X and Linux – except for CDC/ACM mode, which requires a driver information file on Windows (32- and 64-bit drivers available)
- 13.56 MHz HF RFID, supports all ISO 14443A and ISO 15693 tags, with ISO 14443B and ISO 18000-3 available upon request
- Supported tags include Texas Instruments Tag-It HF-I, NXP i-code SLI, NXP Mifare Ultralight, NXP Mifare DESFire, NXP Mifare DESFire EV1
- Extensible and future-proof design: Firmware updates via USB
- USB 2.0 full-speed device, bus-powered, 90mA in active mode, 30mA in stand-by. Thus, can be plugged into any USB port, even into passive hub ports, such as those integrated into keyboards. Less than 100mA when both LEDs are on (Desktop Reader Pro only), and additional 100mA for a total of 200mA when sound output is active (Desktop Reader Pro only)
- 70mW output power, approximately 10cm reading range with typical antenna/tag combinations
- Supported by ubisys® RFID Control Center, a sophisticated graphical user interface application for evaluation, testing and firmware updates (Windows only)
- Three device classes available: USB stick, Desktop Reader Pro with two LEDs (red, green) and a buzzer in premium design, and a classic desktop reader case
- Flexible antenna configurations:
    - 50 Ohm SMA connector for industrial and commercial environments
    - Standard printed circuit-board antenna coils available matched to 50 Ohm at 13.56 MHz
- Customizable OEM versions available, including customer-specific antenna coil designs
- Industrial temperature range: -40 °C ... +85 °C (Desktop Pro: -40°C ... +60°C)
- Different firmware types are available, can be used and exchanged depending on customer requirements

    - **CDC/ACM – Virtual Serial Port**
      The reader appears as a virtual serial port (COM port) by implementing the USB Communications Device Class, Abstract Control Model (CDC/ACM) specification. All major operating systems, including Windows, MacOS X and Linux support such devices out-of-the-box – without the need for special device drivers. An "AT"-like command set is provided to inventory tags and to access them. CDC/ACM was designed for industrial grade applications. The firmware includes support for anti-collision inventory scan (bulk mode) and link-quality assessment based on received signal strength indications.

    - **HID – Virtual USB Keyboard**
      The device appears as a keyboard by implementing the USB Human Interface Device Class (HID) specification with the keyboard usage page. All major operating systems, including Windows, MacOS X and Linux support such devices out-of-the-box – without the need for special device drivers.
      Whenever a tag is identified, it automatically "types" the unique identification number as a hexadecimal number, followed by return. Customized versions, which type additional or customized data (e.g. reader serial number combined with UID, memory content, encryption, certificates etc.) are available upon request.
      HID mode perfectly suited for web applications in heterogeneous IT environments ("cloud")

    - **CCID – Smart Card reader**
      The device appears as a Smart Card reader (Chip Card Interface Device, CCID) with one virtual slot. Standard CCID device drivers, provided with all major operating systems, will be used. Supports access to ISO15693 memory cards, identification of ISO15693 non-memory cards, identification of cards according to ISO14443A-3 (Mifare Classic/Mini/Ultralight) and full access to cards according to ISO14443-4 (Mifare DESFire, NXP SmartMX etc.).
      Compatible with many desktop virtualization solutions (including Microsoft and Citrix products)

Reader devices can be ordered with the firmware of choice (CDC/ACM, HID or CCID) and can later be updated to newer firmware releases as well as to other firmware types, by using the ubisys RFID Control Center (see section 8).

4. Supported TAGs

	CDC		HID		CCID			Remarks
	UID	R/W	UID	R	UID	R/W	D/E	
ISO15693								
ICODE SLI	✓	✓	✓	○	✓	✓	✓	
ICODE SLI-L	✓	✓	✓	○	✓	✓	✓	
ICODE SLI-S	✓	✓	✓	○	✓	✓	✓	
ICODE SLI-SY	✓	✓	✓	○	✓	✓	✓	
ICODE UID	✓	○	✓	○	✓	✓	✓	
ICODE UID-OPT	✓	○	✓	○	✓	✓	✓	
ICODE EPC	✓	○	✓	○	✓	✓	✓	
ICODE 1	-	-	-	-	-	-	-	
ICODE HC	✓	✓	✓	○	✓	✓	✓	
TAG it HF-I	✓	✓	✓	○	✓	✓	✓	
TAG it HF-I Pro	✓	✓	✓	○	✓	✓	✓	
TAG it HF-I Plus	✓	✓	✓	○	✓	✓	✓	
Legic Advant MV	✓	-	✓	-	✓	-	-	
STM LRI64	○	○	○	○	✓	✓	✓	LRI64 tags require 10% modulation depth, 1-out-of-4 coding, single subcarrier setting and high data rate. The CCID firmware can be configured for these parameters. CDC does not currently support configuration. A modified firmware can be provided on request.
STM LRI2K	✓	✓	✓	○	✓	✓	✓	
STM LRIS2K	✓	✓	✓	○	✓	✓	✓	
Fujitsu MB89R118	✓	✓	✓	○	✓	✓	✓	Requires CDC Version 1.16 or above, CCID Version 1.02 or above
ISO14443A storage cards								
Mifare Classic	○	-	✓	-	✓	-	-	
Mifare Ultralight	○	○	✓	○	✓	○	-	
Mifare Plus	○	○	✓	○	✓	○	-	
Mifare Mini	○	○	✓	○	✓	○	-	
ISO14443A-4 processor cards								
Mifare DESFire Mifare DESFire EV1	○	-	✓	-	✓	-	✓	No direct read/write access. Library implementing the application-layer cryptographic protocol to allow reading/writing data is available upon request
Generic ISO14443A-4 cards (e.g. NXP SmartMX)	○	-	✓	-	✓	-	✓	Application-layer protocol depends on the implemented application.

✓ Supported
\- Not supported, or not applicable
○ Available on request as a customer-specific feature

UID read unique identifier (UID)
R/W read/write access
R read-only access
D/E transparent data exchange

Remarks:

Mifare DESFire cards utilize a complex but flexible cryptographic protocol and require authentication, session setup etc. and offer optional data encryption or message integrity checking. The protocol is not implemented in the reader itself, but a library is available on request, which implements the protocol and allows read/write access via the CCID firmware using the transparent data exchange.

Transparent data exchange for ISO15693 tags allows implementing features exceeding the reader-supported read/write access in the host software. A library implementing further features is available upon request.

Notice that ISO 15693 tags with eight-byte data blocks (e.g. Fujitsu MB89R118) require at least CDC version 1.16 or CCID version 1.02 to read data blocks. Previous firmware versions were affected by a bug. The bug did not affect writing.

5. CDC Firmware

The device appears as a virtual serial port (COM port) by implementing the USB Communications Device Class, Abstract Control Model (CDC/ACM) specification. All major operating systems, including Windows (see below), Linux, and MacOS support such devices out-of-the-box.

Microsoft Windows requires an ".inf" file to recognize the device, which installs the built-in Windows driver. This ".inf" file is (currently) not WHQL certified. Therefore, Microsoft Windows displays a warning. If you need WHQL certified drivers for Microsoft Windows please file a feature request (support@ubisys.de). Notice that the ".inf" file is digitally signed however, and thus can be installed on all 64-bit versions of Microsoft Windows.

A simple AT command set is provided to inventory tags and to access them.

5.1. Features

- The virtual serial port allows easy interfacing to customer-specific applications
- Supports ISO15693 (13.56 MHz) tags, e.g. TI Tag-It HF, NXP i-code SLI etc.
- Features a simple to use, "AT-like" command set that allows scanning for tags, reading and writing on-tag memories, modifying physical layer parameters, etc.
- Thin abstraction layer over the ISO15693 protocol layer.

5.2. CDC/ACM AT Command set

5.2.1. Port Settings

The device represents itself as a virtual COM port. As such, port settings, like communication speed, parity or stop bit settings will be ignored. No specific configuration is necessary and the port can be operated at any settings (without influencing the communication speed).

5.2.2. General Frame Format

Request frames are terminated with a carriage-return character (ASCII code 13 decimal; 0x0d hexadecimal; "\r" as C string literal). The device terminates response frames with carriage-return followed by line-feed (ASCII codes 13, 10 decimal; 0x0d, 0x0a hexadecimal, "\r\n" as C string literal). Do not add any white space (space, back-space, tab, carriage-return, line-feed, etc.), it will not be removed by the command-line parser and thus leads to "ERROR" responses.

5.2.3. Device Information

This command queries the product information and firmware version of the device.

Syntax

```
Request              ATI↵

Response             <product description>
                     S/N <serial number>
                     OK
```

Example

```
Request              ATI↵

Response             UBISYS RFID 13.56 MHz (CDC) 1.05 Aug 18 2010
                     S/N 000000A003
                     OK
```

5.2.4. Echo Control

This command enables or disables the echo of typed (received) characters. Enabling the echo is mainly useful for debugging and for manual testing.

Syntax

```
Request              ATE0↵       Disable echo
                     ATE1↵       Enable echo. All received characters will be echoed
back.

Response             OK
```

5.2.5. Light-Emitting Diode Control

This command controls the state of the light-emitting diodes (LEDs). Notice that not all hardware variants feature LEDs. Requires CDC/ACM firmware version 1.12 or above.

Syntax

```
Request              AT+D<led no>=[0|1|2]↵    Changes the state of the specified LED to
                                              0 = off, 1 = on, or 2 = blinking. If no
                                              state is specified, the application
                                              relinquishes control to the firmware.

Response             OK
```

Example

```
Request              AT+D1=1↵                 Turns LED number 1 on.
Response             OK

Request              AT+D2=0↵                 Turns LED number 2 off.
Response             OK

Request              AT+D3=2↵                 Makes LED number 3 blink.
Response             OK

Request              AT+D3=↵                  Returns control of LED3 to the firmware.
```

Response	OK	

5.2.6. Buzzer Control

This command controls the operating mode of the on-board buzzer. Notice that not all hardware variants feature a buzzer. Requires CDC/ACM firmware version 1.13 or above.

Syntax

Request	AT+B=[0|1]↵	*Changes the operating mode of the buzzer to 0 = off, or 1 = on. If no operating mode is specified, the application relinquishes control to the firmware (a short beep whenever a transponder is detected).*
Response	OK	

Example

Request	AT+B=1↵	*Turns buzzer on.*
Response	OK	
Request	AT+B=0↵	*Turns buzzer off.*
Response	OK	
Request	AT+B=↵	*Returns control of the buzzer to the firmware.*
Response	OK	

5.2.7. Inventory Scan

Requests an inventory scan. Returns a list of 64-bit unique identifiers and the received signal strength indicators, as reported by main and auxiliary receivers. If no tags were found, the response is OK (no UIDs).

ISO 15693 command: 01h (mandatory, supported by all tags)

Syntax

```
Request              AT+I↵

Response             +UID=<uid>,+RSSI=<main>/<aux>     [possibly repeated or absent]
                     OK

                     ERROR      automatic scan mode (section 0) enabled
```

Example

```
Request              AT+I↵

Response             +UID=E00402000058913D,+RSSI=6/6
                     +UID=E00402000023F23C,+RSSI=3/2
                     OK
```

5.2.8. Inventory Scan (without Anti-Collision)

Requests an inventory scan without anti-collision. Returns the 64-bit unique identifier of the identified tag and the received signal strength indicators, as reported by main and auxiliary receivers. If no tag was found, the response is OK (no UID).

ISO 15693 command: 01h (mandatory, supported by all tags)

Requires CDC/ACM firmware version 1.09 or above.

Syntax

```
Request              AT+i↵

Response             +UID=<uid>,+RSSI=<main>/<aux>      [may be absent]
                     OK

                     ERROR      automatic scan mode (section 0) enabled
```

Example

```
Request              AT+i↵

Response             +UID=E00402000058913D,+RSSI=6/6
                     OK
```

5.2.9. Select Tag

Select a single tag for further processing. The tag access commands (get system information, read data block, write data block) can either address all tags in the field or only a single tag. If it is known or expected that more than one tag is in the field, a single tag should be selected.

Requires CDC/ACM firmware version 1.09 or above.

Syntax

```
Request          AT+SELECT=<uid>↵      Select the specified tag for further processing.

                 AT+SELECT=↵           Deselect tag. Address all tags in the field from
                                       now on.
Response         OK
```

Example

```
Request          AT+SELECT=E00402000058913D↵

Response         OK

(tag access commands address the specified tag)

Request          AT+SELECT=↵

Response         OK

(tag access commands address all tags in the field)
```

5.2.10. Get System Information

Obtain the tag's system information. Returns 64-bit unique identifier, data storage format identifier (DSFID), application family identifier (AFI), memory block count (BC) and size (BS), and IC manufacturer defined data (IC). Notice that not all tags support this command and not all tags support all items. DSFID, AFI and IC are displayed in hexadecimal format, BC and BS in decimal format.

ISO 15693 command: 2Bh (optional, supported by most tags, including NXP i-code SL-2, TI Tag-It HF-I)

Syntax

```
Request             AT+S↵

Response            +UID=<uid>,DSFID=<dfsid>,AFI=<afi>,BC=<bc>,BS=<bs>,IC=<ic>
                    OK

                    ERROR       tag reported an error, did not respond or automatic scan
                                mode (section 0) enabled
```

Please note that the presence of the DFSID, AFI, BC, BS and IC fields depends on what the tag supports.

Examples

```
Request             AT+S↵

Response            +UID=E00402000058913D,DSFID=00,AFI=01,BC=40,BS=4,IC=02
                    OK
```

DFSID 00h, AFI 01h, 64 blocks of 4 bytes.

```
Request             AT+S↵

Response            +UID=E00402000058913D,BC=100,BS=1
                    OK
```

No DFSID, no AFI, 256 blocks of 1 bytes.

```
Request             AT+S↵

Response            +UID=E00402000058913D
                    OK
```

No memory-related capabilities reported.

5.2.11. Read Data Block

Read data block from tag memory. The block number must be within the range supported by the tag, i.e. the within range returned by the “Get System Information” request (BC, if command is supported). Returns the length of the data block, followed by the actual data in binary format (separated by a colon).

ISO 15693 command: 20h (optional, supported by most tags with on-chip memory, including NXP i-code SL-2, TI Tag-It HF-I)

Syntax

```
Request             AT+R<block number>↵

Response            +DATA <number of raw bytes>:<raw data bytes>
                    OK

                    ERROR  tag reported an error or did not respond, or automatic scan
                           mode (section 0) enabled
```

Example

```
Request             AT+R0↵

Response            +DATA 4:1234
                    OK

Request             AT+R88↵

Response            ERROR  (e.g. if address is out of range)
```

5.2.12. Write Data Block

Write data block to tag memory. The block number must be within the range supported by the tag, i.e. within the range returned by the “Get System Information” request (BC, if command is supported). The Length must match the size of a data block as returned in the BS parameter.

ISO 15693 command: 21 (write single block, optional)

Notice: There are two variants of the write single block command. The first variant (lower-case “w”) has

the option request flag clear, the second variant (upper-case “W”) has the option request flag set. It depends on the tag, which command variant is supported. If you encounter errors or the memory block content does not change, try the alternate option flag setting.

The data portion is in raw binary format, i.e. all values in the range 0-255 are allowed and written to the tag without modification.

Syntax

Request	AT+w<block number>, <size>↵ <data>	*without option flag*
Request	AT+W<block number>, <size>↵ <data>	*with option flag*
Response	OK	
	ERROR	*tag reported an error or did not respond, or automatic scan mode (section 0) enabled*

Examples

Request	AT+w0,4↵ 1234	Writes 0x31 0x32 0x33 0x34 to memory block 0, option flag clear
Response	OK	
Request	AT+W23,4↵ Text	Writes 0x54 0x65 0x78 0x74 to memory block 23, option flag set
Response	OK	

5.2.13. Lock Data Block

Lock data block in tag memory. Requires CDC/ACM firmware version 1.15 or above. The block number must be within the range supported by the tag, i.e. the within range returned by the “Get System Information” request (BC, if command is supported).

ISO 15693 command: 22h (optional, supported by most tags with on-chip memory, including NXP i-code SL-2, TI Tag-It HF-I)

Notice: There are two variants of the lock block command. The first variant (lower-case “l”) has the option request flag clear, the second variant (upper-case “L”) has the option request flag set. It depends on the tag, which command variant is supported. If you encounter errors, try the alternate option flag setting.

ATTENTION: Once a block has been locked, it cannot be unlocked any more. The contents of the block will be permanently write-protected, i.e. subsequent write attempts to the same block will fail. Other blocks are not affected.

Syntax

Request	AT+l<block number>↵	*without option flag*
Request	AT+L<block number>↵	*with option flag*
Response	OK	
	ERROR *tag reported an error or did not respond, or automatic scan mode (section 0) enabled*	

Examples

Request	AT+w0,4↵ 1234	Writes 0x31 0x32 0x33 0x34 to memory block 0, option flag clear
Response	OK	
Request	AT+l0↵	Permanently write-protects block 0
Response	OK	

5.2.14. Enable/Disable RF field

Syntax

```
Request             AT+RF=<0|1>↵

Response            OK
```

Example

```
Request             AT+RF=0↵      Disable RF field

Response            OK


Request             AT+RF=1↵      Enable RF field

Response            OK
```

5.2.15. Query RF field status

Syntax

```
Request             AT+RF?↵

Response            +RF=<0|1>
                    OK
```

Example

```
Request             AT+RF?↵

Response            +RF=1
                    OK
```

5.2.16. Automatic Scan Mode

Enable or disable the automatic scan mode. In the automatic scan mode, indications are created automatically whenever a tag enters or leaves the field.

NOTICE: While in any of the scan modes, all other commands are not available. You need to quit scan mode first, using AT+SCAN0.

Syntax

```
Request             AT+SCAN<0|1|2>↵      0  disable automatic scan mode
                                         1  enable automatic scan mode
                                         2  enable automatic scan mode with extended
                                            scan indications (including RSSI)

Response            OK

Indications         SCAN:+UID=<uid>                          Tag found (new tag)
                    SCAN:-UID=<uid>                          Tag lost
Mode 2 only:        SCAN:+UID=<uid>,+RSSI=<main>/<aux>       New tag or RSSI changed
```

Example

```
Request             AT+SCAN1↵                                        Enable scan mode 1

Response            OK

(from this point forward the reader only accepts AT+SCAN commands, in particular AT+SCAN0
to leave scan mode)

Indications         SCAN:+UID=E00402000058913D                       Tag found
                    SCAN:-UID=E00402000058913D                       Tag lost

Request             AT+SCAN2↵                                        Enable scan mode 2

Response            OK

(from this point forward the reader only accepts AT+SCAN commands, in particular AT+SCAN0
to leave scan mode)

Indications         SCAN:+UID=E00402000058913D,RSSI=7/7              Tag found
                    SCAN:+UID=E00402000058913D,RSSI=6/5              RSSI changed
                    SCAN:-UID=E00402000058913D                       Tag lost

Request             AT+SCAN0↵                                        Scan mode disabled

Response            OK

(no indications generated anymore, you may use other commands again)
```

6. HID Firmware

The HID firmware acts as a USB HID keyboard and “types” the UID of a found tag by using virtual keystrokes. As such, it is possible to use the reader in conjunction with any application, without modifications.

6.1. Features

- No dedicated device driver required. All major operating systems (Microsoft Windows, Mac OS X, Linux) ship with a driver, which will be used.
- Whenever a tag is identified, it automatically “types” the unique identification number as a hexadecimal number, followed by return.
- Supported standards are ISO15693 and ISO14443A (integrated “multi-ISO” firmware and dedicated firmware images for each standard)
- Extended firmware versions available that type the reader’s serial number in front of the tag’s UID. This version allows you to know with which reader (i.e. where) the tag has been scanned.
- Custom configurations can be provided, which output tag memory contents or enter customer-defined prolog/epilog key sequences. Please contact ubisys® support for details.

6.2. Operation

Whenever a tag is detected first, its UID is typed via emulated keystrokes. As long as the tag is in the field, no further action is taken. If the tag leaves the field and enters it again, the UID is typed again.

With the CCID Firmware, the device appears to the host operating system as a single-slot Smart Card Reader. If a tag is in the RF field of the reader, the reader reports an ”inserted card”. If there is no tag in the field, this is reported as “card ejected”.

The device itself complies with the USB 2.0 CCID 1.1 specification (chip card interface device) to allow the host computer to use a standard device driver, shipped with all major operating systems.

Applications use the operating-system-provided Smart Card API to access the reader and the tags. A native Smart Card API is available on Microsoft Windows 2000 and above. The same API has been adopted on MacOS X and Linux (pcsc-lite).

The reader emulates the T=1 APDU protocol exchange, allowing the host computer to exchange commands and data with the reader and/or tag by sending and receiving APDUs.

For ISO 14443-4, which is based on APDUs natively, the reader establishes a transparent channel and forwards APDUs between host and tag.

For memory cards (ISO15693), an APDU emulation layer is provided, which translates certain predefined APDUs to appropriate ISO 15693 commands. Besides the APDU emulation layer, a transparent mode can be enabled for ISO 15693, allowing the host application to communicate directly with the tag. This feature allows customers to operate any tag that complies with the ISO 15693 standard. In particular, application software may use any manufacturer-specific “custom command” codes.

A set of APDUs is defined to read the UID of the tag in the field and to configure the reader. These APDUs will always be processed by the reader itself.

7.1. Driver Installation

On Microsoft Windows systems, the driver will be installed automatically. On Windows versions that originally shipped without native CCID support, like Windows 2000 and Windows XP, you might need to allow Windows Update to be searched for a suitable device driver. If you are using Windows 2000, make sure that Service Pack 3 is installed.

Linux support requires that the packages pcsc-lite and libccid are installed.[1]

MacOS X Leopard, Snow Leopard and Lion also include versions of pcsc-lite.

Your ubisys product is automatically supported on systems with libccid version 1.4.5 as of October 11, 2011 and above. Depending on your OS type and version, the ubisys USB product and vendor ID (0x19a6:0x0009) might not be included in shipped configurations yet and must be entered in a configuration file belonging to PC/SC Lite in this case. Under MacOS X, the file is

/usr/libexec/SmartCardServices/drivers/ifd-ccid.bundle/Contents/Info.plist

[1] The exact package names depend on the distribution used.

Under Linux, this file is usually

/etc/libccid_Info.plist

However, this is only a convention and might to apply to your particular Linux distribution.

The file is an Apple Property List in XML format and must be extended in the following way:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>

       [....]

       <key>ifdVendorID</key>
       <array>
              [....]
|             <string>0x19a6</string>
       </array>

       <key>ifdProductID</key>
       <array>
              [....]
|             <string>0x0009</string>
       </array>

       <key>ifdFriendlyName</key>
       <array>
              [....]
|             <string>ubisys 13.56 MHz RFID (CCID)</string>
       </array>

       [....]

</dict>
</plist>
```

Please restart the PC/SC service or reboot your machine after having modified the file. Also note that you need elevated access rights to modify the file. Use sudo on MacOS and Linux to obtain elevated access rights.

To verify correct operation on MacOS, open a terminal shell and enter pcsctest, which will run the MUSCLE PC/SC Lite Test Program. Your ubisys reader should be listed. When prompted to enter the reader number, enter the number next to “ubisys 13.56 MHz RFID (CCID)”. You may then present a tag to the reader in order to see the tag’s ATR.

To verify correct operation on Linux, open a terminal shell and enter pcsc_scan. If this program is not available, install the pcsc-tools package, first (e.g. “sudo apt-get install pcsc-tools” on ubuntu, “emerge pcsc-tools” on gentoo). You might also use OpenSC for verification (“sudo apt-get install opensc”, then “opensc-tool -l" to list the reader). Finally, it is also possible to run pcscd in foreground mode with debugging output: “pcscd -f -d” in a terminal shell will also generate output when a tag is presented to the reader.

For example, the following output was created on a PowerMac G5 running Gentoo Linux and 64-bit PowerPC userland while an NXP i-code SL2 card was presented to the reader:

```
ubisys-macg5 ~ # pcsc_scan
PC/SC device scanner
V 1.4.18 (c) 2001-2011, Ludovic Rousseau <ludovic.rousseau@free.fr>
Compiled with PC/SC lite version: 1.7.2
Using reader plug'n play mechanism
Scanning present readers...
0: ubisys 13.56 MHz RFID (CCID) (00000002F2) 00 00

Fri Mar 23 16:37:23 2012
Reader 0: ubisys 13.56 MHz RFID (CCID) (00000002F2) 00 00
  Card state: Card inserted,
  ATR: 3B 8F 80 01 80 4F 0C A0 00 00 03 06 13 00 35 00 00 00 00 4E

ATR: 3B 8F 80 01 80 4F 0C A0 00 00 03 06 13 00 35 00 00 00 00 4E
+ TS = 3B --> Direct Convention
+ T0 = 8F, Y(1): 1000, K: 15 (historical bytes)
  TD(1) = 80 --> Y(i+1) = 1000, Protocol T = 0
-----
  TD(2) = 01 --> Y(i+1) = 0000, Protocol T = 1
-----
+ Historical bytes: 80 4F 0C A0 00 00 03 06 13 00 35 00 00 00 00
  Category indicator byte: 80 (compact TLV data object)
    Tag: 4, len: F (initial access data)
      Initial access data: 0C A0 00 00 03 06 13 00 35 00 00 00 00
+ TCK = 4E (correct checksum)

Possibly identified card (using /usr/share/pcsc/smartcard_list.txt):
3B 8F 80 01 80 4F 0C A0 00 00 03 06 13 00 35 00 00 00 00 4E
3B 8F 80 01 80 4F 0C A0 00 00 03 06 .. 00 35 00 00 00 00 ..
        i-Code SL2 (as per PCSC std part3)
```

7.2. Features and Tag Support

The CCID firmware supports the following standards:

Tag/Card Type	UID	Read/Write (through built-in functions)	Transparent access
ISO15693	✓	✓	✓
ISO14443-3 (Type A)	✓	-	-
ISO14443-4 (Type A)	✓	-	✓

ISO15693 tags are accessible through the built-in function “Read Binary” and “Update Binary” for straight-forward memory-card operations. For application-specific/non-standard commands, a transparent mode is available.

ISO14443-3 (Type A) cards include the family of proprietary Philips/NXP Mifare Classic and Ultralight cards and are currently supported in a very limited way only. Their presence is detected and the UID can be read.

ISO14443-4 cards are fully fledged processor cards, which are accessible through a vendor-defined protocol, which will be exchanged transparently and implemented in the host application. For example, NXP Mifare DESFire cards belong to this class of tags. Libraries implementing the communication protocols are available on request to qualified customers. Please contact support@ubisys.de for details.

7.3. ATR Strings

A classic smart card sends an “answer to reset” through the smart card reader to the host operating system upon completion of its reset sequence. This behaviour is emulated for the contactless system.

An ATR is generated according to the PC/SC Interoperability Specification for ICCs and Personal Computer Systems, Part 3. Requirements for PC-Connected Interface, section 3.1.3.2.3.

Notice that ATR response strings do not contain the UID, i.e. cards to the same type produce the same ATR. Use the Get UID command if you are interested in the tag’s UID.

7.3.1. ATR for ISO14443-4

The ATR for an ISO14443-4 card is built around the historical bytes, as received from the tag itself as the answer to the RATS command. Up to 15 historical bytes are included in the ATR.

Byte	Designation	Value
0	Initial Header	3B
1	T_0	8*n*
2	T_{D1}	80
3	T_{D2}	01
4..(3+n)	Historical bytes $T_1..T_{n+1}$	
4+n	T_{CK} (Check byte)	

7.3.2. ATR for Memory Cards (ISO15693, ISO14443-3 Type A)

Byte	Designation	Value
0	Initial Header	3B
1	T_0	8F
2	T_{D1}	80
3	T_{D2}	01
4	Category indicator	80
5	Application identifier presence indicator	4F
6	Length	0C
7..11	Registered Application Provider (PC/SC)	A0 00 00 03 06
12	Standard	SS
13..14	Card name	NN NN
15..18	Reserved	00 00 00 00
19	T_{CK} (Check byte)	

Standard

Standard	SS
ISO14443-3	03
ISO15693 (-3)	0B

Card name

Card	NN NN
Undefined/unknown	00 00
Mifare Standard 1K	00 01
Mifare Standard 4K	00 02
Mifare Ultra light	00 03
TI Tag It HF	00 12
NXP i-code SLI	00 14
NXP i-code SL2	00 35

7.4. APDU Format

The reader supports short APDUs according to ISO7816-4. Requests consist of a fixed header and a variable payload of up to 255 bytes. Responses consist of a variable payload of up to 256 bytes, followed by two status bytes.

7.4.1. Request APDU

Class CLA	Instruction INS	Parameter 1 P1	Parameter 2 P2	L_c	Payload	L_e
1 byte	1 byte	1 byte	1 byte	1 byte	variable	1 byte
Mandatory				Optional		

The first four fields (CLA, INS, P1, P2) are mandatory. Each of them is encoded in one byte.

The field L_c determines the payload length. The field is absent if the payload length is 0.

The field L_e determines the maximum number of bytes expected in the response payload field. If no payload is expected (0 bytes), this field is absent. 256 bytes of expected response payload is encoded as 00h.

7.4.2. Response APDU

Payload	SW1	SW2
variable	1 byte	1 byte
Optional	Mandatory	

The payload field is optional, the status bytes SW1 and SW1 are always present.

7.5. APDU Definitions

7.5.1. Tag Identification Commands

7.5.1.1. Get Data Command

Standard: PC/SC: Interoperability Specification for ICCs and Personal Computer Systems, Part 3, Section 3.2.2.1.3.

Request

Command	CLA	INS	P_1	P_2	L_e
Get UID	FFh	CAh	00h	00h	Expected length of UID
Get all historical bytes (ISO14443, excl. CRC)	FFh	CAh	00h	00h	Expected length of historical bytes

Response

Data	SW1	SW2

Response codes

SW1	SW2	Meaning
90h	00h	The complete data is returned.
62h	82h	The complete data is returned. End of data reached, i.e. Le is larger than the length of the UID/historical bytes.
6Ch		Wrong length requested, i.e. Le is less than the length of the UID/historical bytes. SW2 indicates the correct length.
6Ah	81h	Function not supported.

7.5.2. Configuration commands

Valid for: all standards, independent of active tag

7.5.2.1. Get Configuration

Request

CLA	INS	P_1	P_2	L_e
FFh	33h	00h	Configuration item	ignored

Response

Configuration value	SW1	SW2

Please note that Le is ignored. The stored value is returned without being truncated, independently of Le.

Response codes

SW1	SW1	Meaning
90	00	Command executed successfully. The configuration item is delivered in the payload.
6B	00	Unsupported parameter P1-P2 combination.

The following configuration items are supported:

Configuration item [P2]	Description	Length	Values
80h	ISO15693 modulation depth	1 byte	00h – 10% modulation depth 01h – 100% modulation depth (OOK)
81h	ISO15693 data-rate	1 byte	00h – low data rate 01h – high data rate
82h	ISO15693 subcarrier setting	1 byte	00h – single subcarrier 01h – double subcarrier
8eh	ISO15693 default memory configuration[2]	2 byte	See table below
8fh	Clear ISO15693 default memory configuration (write-only)	0	-

The ISO15693 data rates are as following:

Data rate	Single Subcarrier	Dual Subcarrier
Low	6.62 kbits/s	6.67 kbits/s
High	26.48 kbits/s	26.69 kbits/s

Parameter layout for the ISO15693 default memory configuration

Number of blocks (minus 1)	Block size in bytes (minus 1)
1 byte	1 byte

Both values are encoded as the actual value minus 1. This allows encoding values of 1 to 256 for both fields.

If the default configuration is read back and no default configuration was previously set, an empty response is returned.

[2] This configuration is used if the tag does not support the ISO15693 “Get System Information” command, which, among other information, returns the memory layout of the tag.

7.5.2.2. Set Configuration

Set a configuration option. Please not that only the current configuration is updated. The configuration must be explicitly saved to non-volatile memory to be preserved permanently. Otherwise, the previously saved configuration will be restored on the next system startup.

Request

CLA	INS	P1	P2	Lc		Le
FFh	33h	01h	Configuration item	Length of the value to set	Value to set	ignored

Response

SW1	SW2

Response codes

SW1	SW1	Meaning
90h	00h	Configuration updated.
67h	00h	Wrong length.
6Ah	81h	Function not supported. The specified parameter value is invalid
6Bh	00h	Unsupported parameter P1-P2 combination.

7.5.2.3. Save Configuration

Save the configuration to non-volatile memory. The current configuration will be preserved and reloaded on the next system start-up.

Request

CLA	INS	P1	P2	Le
FFh	33h	80h	43h	ignored

Response

SW1	SW2

Response codes

SW1	SW1	Meaning
90h	00h	Command executed successfully. The configuration was saved.
64h	00h	Failed to save configuration to non-volatile memory.

7.5.3. Memory Card Access Commands

Standard: PC/SC: Interoperability Specification for ICCs and Personal Computer Systems, Part 3

Supported tags: ISO15693 memory tags. All memory tags are supported, which implement the “Read Single Block” or “Write Single Block” commands, respectively.

The memory layout of the tag is automatically detected for tags which support the optional but widely adopted “Get System Information” command. For tags not supporting this command, the (default) memory layout, i.e. the number of blocks and the block size, must be configured manually (section 7.5.2) before the “Read Binary” and “Update Binary” commands can be used. This default memory layout will only be used if the layout cannot be determined automatically.

7.5.3.1. Read Binary

This command reads data from the memory card.

Request

CLA	INS	P1	P2	Le
FFh	B0h	Address MSB	Address LSB	Amount of data to read (in bytes)

Response

Data	SW1	SW2

Response codes

SW1	SW1	Meaning
90h	00h	Command executed successfully and the requested data is returned.
62h	82h	EOF reached. The requested byte range exceeded the memory. Less data than requested is returned.
6Ah	81h	Function not supported. No memory card available or reader has entered transparent mode.
6Ah	82h	File not found. The requested start address exceeds the memory capacity.

7.5.3.2. Update Binary

Request

CLA	INS	P1	P2	Lc	Payload
FFh	D6h	Address MSB	Address LSB	Amount of data to update (in bytes)	Data to update (Lc bytes)

Response

SW1	SW2

Response codes

SW1	SW1	Meaning
90h	00h	Command executed successfully and the data was written.
62h	82h	EOF reached. The requested byte range exceeded the memory. Not all data could be written.
6Ah	81h	Function not supported. No memory card available or reader has entered transparent mode.
6Ah	82h	File not found. The requested start address exceeds the memory capacity.

7.5.4. Extended ISO15693 Information Commands

Request

CLA	INS	P1	P2	Le
FFh	A2h	see table		ignored

P1	P2	Description	Response Payload
00h	00h	Get total capacity in bytes.	2 bytes
00h	01h	Get number of blocks.	1 byte
00h	02h	Get block size in bytes (1- 256 bytes).	2 bytes
01h	00h	Get AFI (Application family identifier)	1 byte
01h	01h	Get DFSID (Data storage format identifier)	1 byte

2-byte values are encoded in little-endian byte order.

Response

Payload	SW1	SW2

Response codes

SW1	SW1	Meaning
90h	00h	Command executed successfully and the requested information is contained in the payload.
6Ah	82h	File not found. The tag did not submit the requested parameter.
6Bh	00h	Wrong parameter P1-P2 submitted.

7.5.5. ISO15693 Transparent mode

7.5.5.1. Enter Transparent Mode

This command enters the ISO15693 transparent mode, which allows exchanging arbitrary commands between the tag and the reader. Two modes of operation are available. In the first mode, the reader does not 'ping' the tag periodically and as such, is not able to detect if the tag is removed from the reader's field. This mode can be used if tags are set into a state where they do not answer (addressed) inventory requests anymore or if a custom inventory sequence should be used.

In the second mode, the tag is pinged periodically by sending an addressed inventory request, which must be answered as long as the tag is in the field and able to communicate. By this means, the removal of the tag from the reader's field can be detected.

Please note that the memory card access commands (section 7.5.3) are not available when using the transparent mode. The extended ISO15693 information commands (section 7.5.4) might deliver outdated information if the periodic ping was disabled and the tag was substituted or removed.

Request

Command	CLA	INS	P1	P2 Ping mode	Le
Enter transparent mode. Do not ping tag while in transparent mode.	FFh	27h	01h	00h	ignored
Enter transparent mode. Ping tag periodically.	FFh	27h	01h	01h	ignored

Response

SW1	SW2

Response codes

SW1	SW1	Meaning
90h	00h	Command executed successfully. Transparent mode activated.
6Ah	81h	Function not supported. Either, transparent mode is already activated or a non-ISO15693 tag is in the field.

7.5.5.2. Leave Transparent Mode

Disables the transparent mode and returns to normal operation.

Request

CLA	INS	P1	P2	Le
FFh	27h	02h	00h	ignored

Response

SW1	SW2

Response codes

SW1	SW1	Meaning
90h	00h	Command executed successfully. Normal mode reactivated.
6Ah	91h	Function not supported. Transparent mode has not been not activated.

7.5.5.3. Transparent Data Exchange

This command exchanges data transparently with the tag. The reader does not interpret the data, simply forwards it to the tag and captures the response.

Two variants exist. The first variant is suitable for most requests. It transmits the request frame and immediately captures the response frame. The second variant transmits the request and sends a “Next Slot” command (end of frame marker) to the tag, approx. 15 ms after the request frame was sent. Afterwards, the response is captured.

Request

Command	CLA	INS	P1	P2	Lc	Payload	Le
Exchange transparent data.	FFh	27h	03h	00h	Payload length	Request data	Maximum response length
Exchange transparent data. Send a “Next Slot” command approx. 15 ms after the frame is transmitted. The response will be captured after the “Next Slot” command was sent.	FFh	27h	03h	01h			

Response

Response	SW1	SW2

Response codes

SW1	SW1	Meaning
90h	00h	Command executed successfully. The response is delivered in the response payload field.
62h	82h	Command executed successfully. The response is delivered in the response payload field. The response length was shorter than the specified Le.
6Ch	FFh	Wrong length, i.e. response buffer overflow. The submitted response length Le was too short.

8. Firmware update

The firmware contained in the device can be updated by using RFID Control Center. This allows programming a new firmware image into the device, i.e. transforming e.g. a CDC device into a CCID device or updating to a more recent firmware version. Firmware updates may be conducted on 32-bit and 64-bit Windows (requires RFID Control Center version 1.4 or above). Qualified customers needing firmware upgrade ability on other platforms should contact the ubisys RFID support team.

The ubisys RFID Control Center software is available at http://www.ubisys.de/rfid/download-software.html free of charge and includes an online help, which explains the necessary steps. The package also includes common firmware images. Notice that customer-specific firmware images are typically delivered via e-mail.

Step 1a (applies to HID devices): To initiate firmware update of a device currently acting as a keyboard (HID firmware), click on Update HID, when the device is attached to your PC. If you have more than one RFID device in HID mode, click on the drop-down arrow to see the list of all devices and hover over an entry to see the serial number.

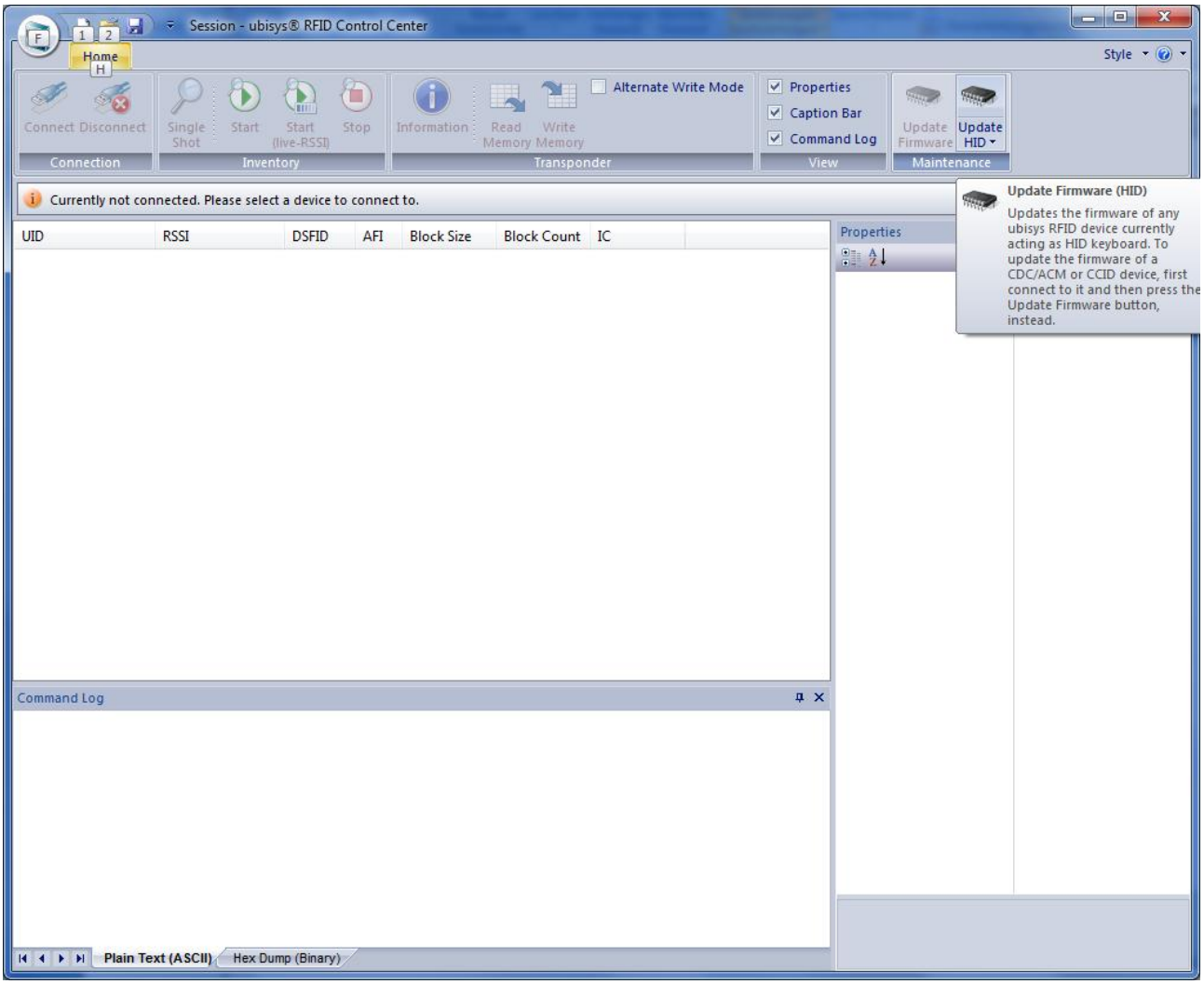

Figure 1: Initiating Firmware Update of HID Devices (Step 1a)

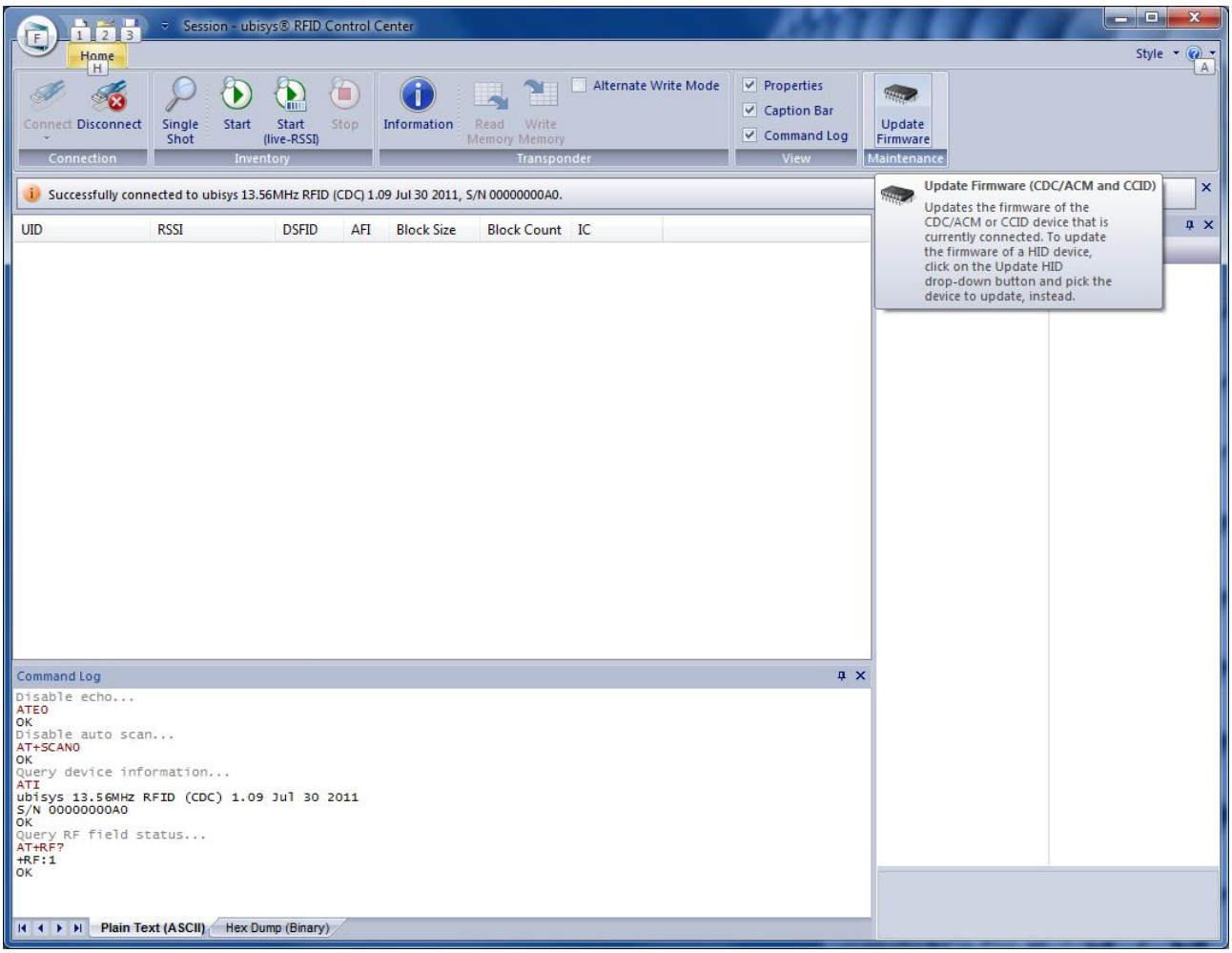

Figure 2: Initiating Firmware Update of CDC and CCID Devices (Step 1b)

Step 1b (applies to CDC/CCID devices): If you are planning to update a CDC or CCID device, first use the connect command to click on the device, which you want to update. Then, click on the Update Firmware command.

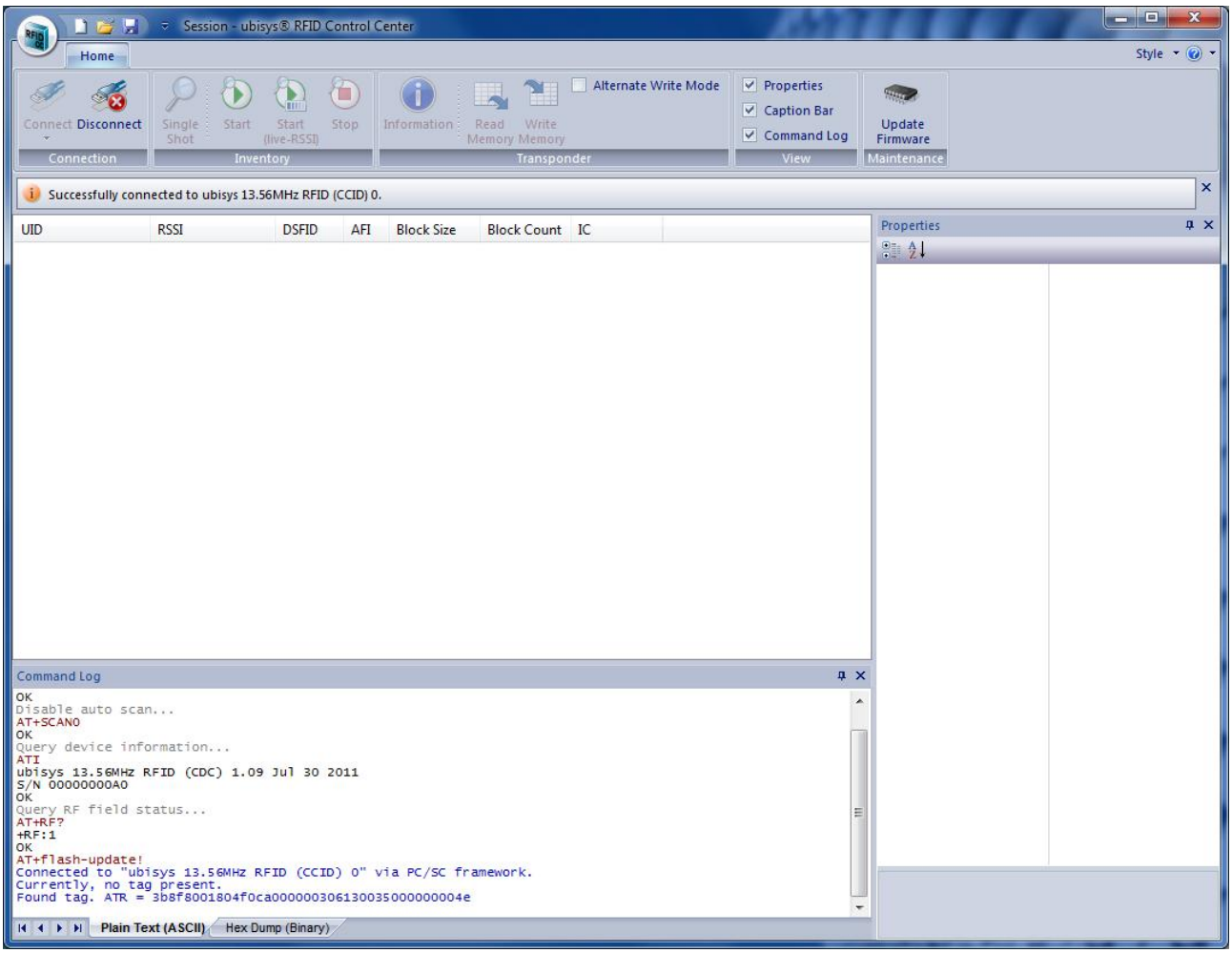

Figure 3: Validating Presence of a Tag in CCID Mode (Step 1b – CCID only)

Notice (applies to CCID devices): In case of CCID devices, there must be a tag in reading range when conducting the firmware update. You can verify this by observing the command log when presenting a card to the reader. The card ATR will be printed for any card that enters reading range. Notice that the ATR does not comprise UID information, i.e. different cards of the same type will create the same ATR. You have to use the Get UID command in your applications to obtain the UID.

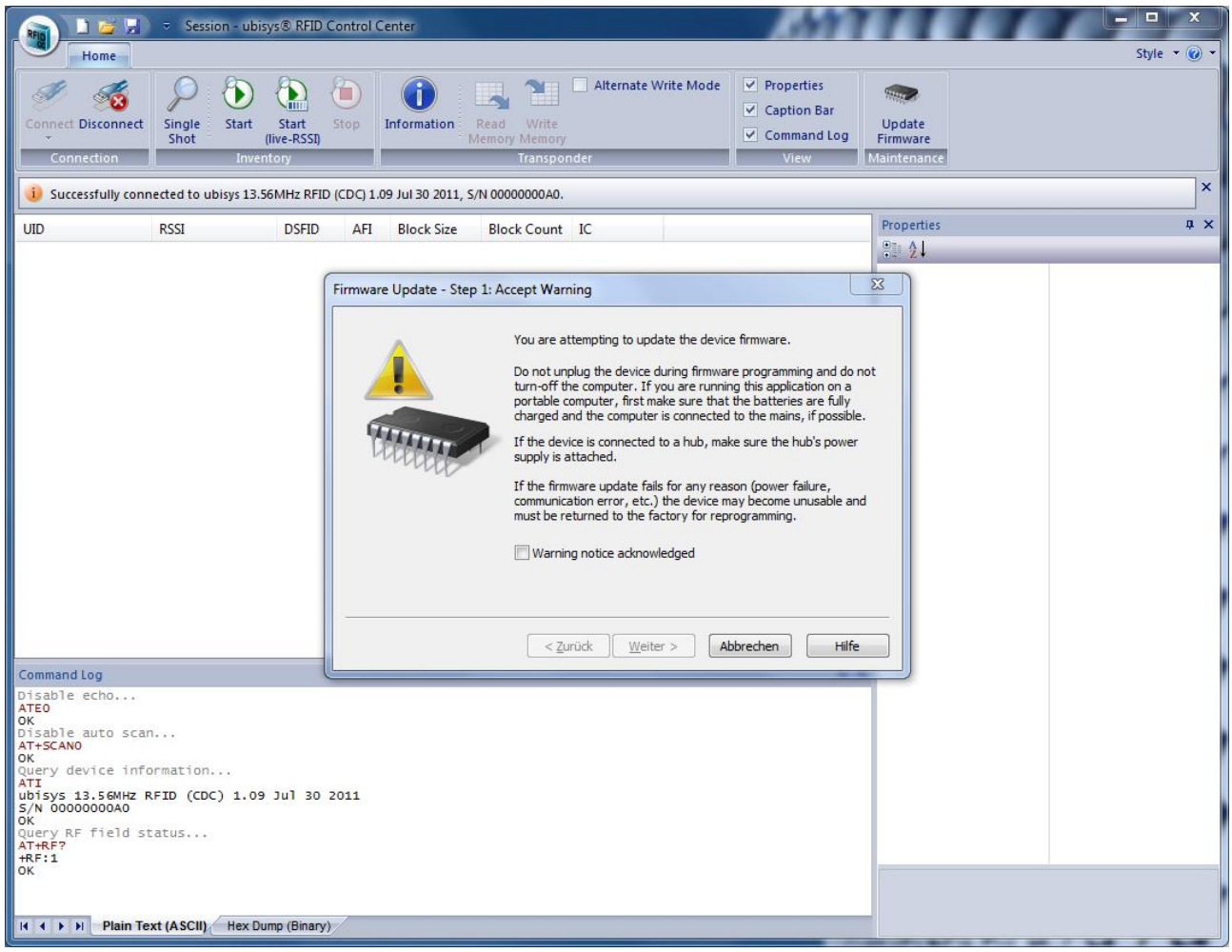

Figure 4: Acknowledging the Warning Notice (Step 2)

Step 2: Acknowledge the warning message.

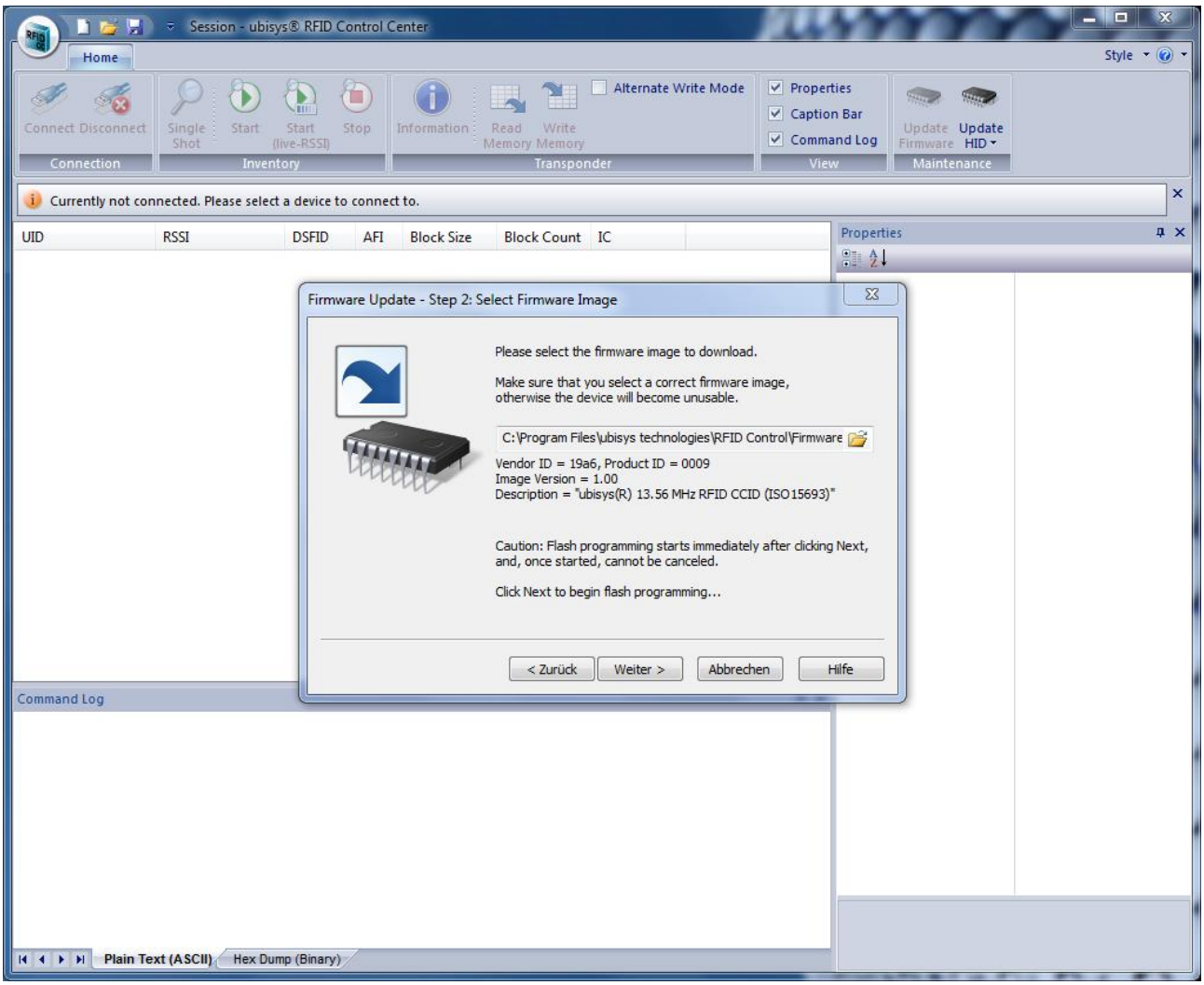

Figure 5: Selecting the Desired Firmware Image

Step 3: Select the new firmware image. All common firmware images can be found in the installation folder of RFID Control Center (typically C:\Program Files\ubisys technologies\RFID Control\Firmware). Make sure that the USB connection is reliable while programming the firmware. Do not turn off intermediate hubs. If you are using a battery powered device for firmware programming, make sure the battery is fully charged and use the power cable if possible. Click Next to start programming.

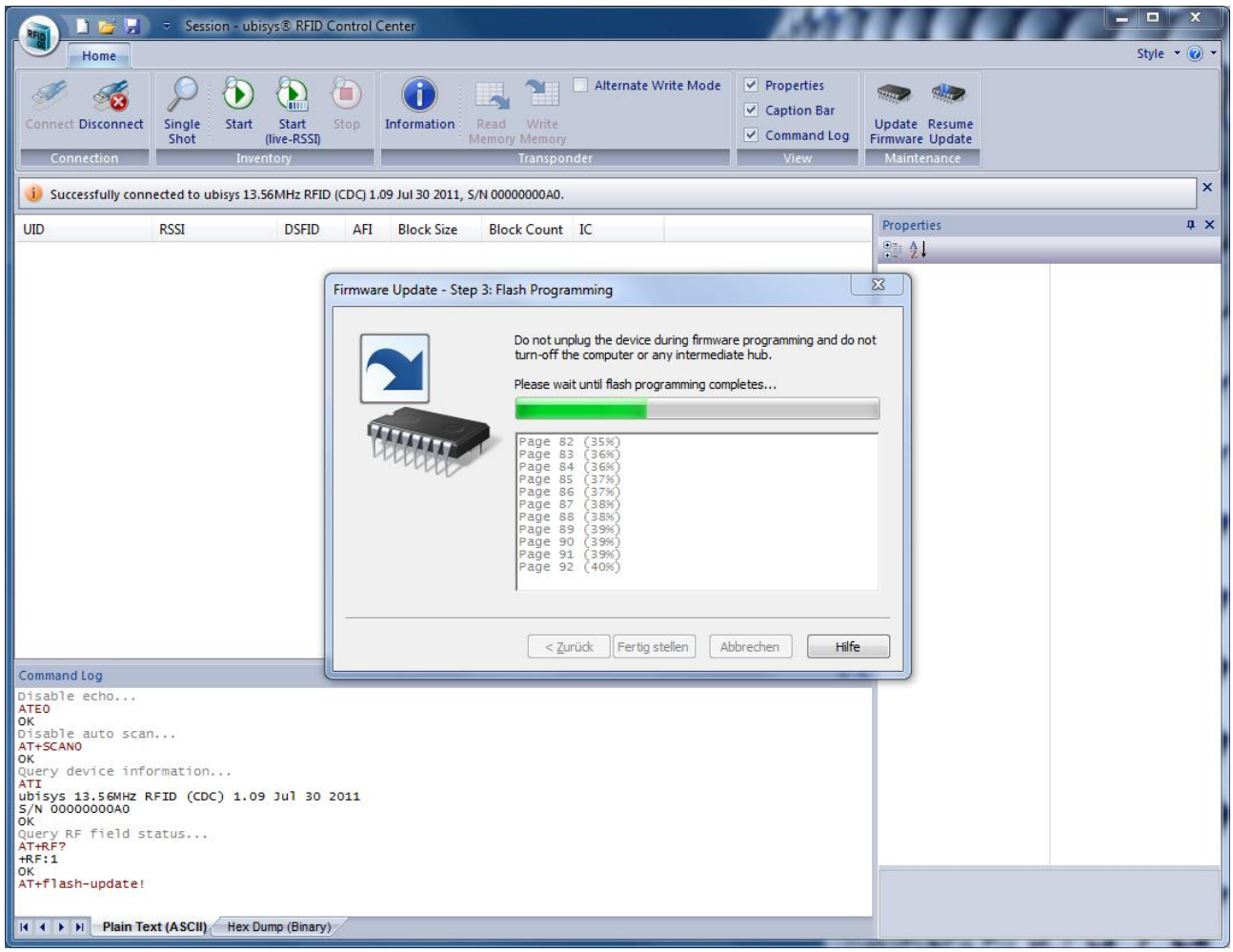

Figure 6: Firmware Update in Progress

Step 4: Wait until the flash programming sequence has successfully completed.

Once the firmware update has successfully completed, the device appears either as HID Keyboard, COM port, or smart card reader – according to its desired USB device role.

Notice: If the update fails for any reason, you might be able to resume the update using the “Resume Update” command. However, this should normally not be necessary. If you cannot use the Resume Update command successfully, detach the device from USB and reattach it. If the Resume Update Button does not appear after reattaching the device, it must be returned to the factory.

ubisys provides a C++ RFID/Smart Card API library, which provides an easy and convenient way of dealing with the Smart Card API. It can be used on Windows, MacOS X and Linux, as all operating systems use the same underlying Smart Card API.

9.1. Overview

9.2. Basic RFID/Smart Card API

The classes CSmartCardContext and CSmartCard encapsulate handles to opaque Smart Card API objects, which are established in the constructors and released in the destructors.

Copying of and assigning to class instances is not supported. To enforce correct usage, the classes provide private copy constructors and assignment operators (operator=), to prevent objects from being copied in inappropriate ways. Class instances shall always be passed by reference.

The same applies to CSmartCardTransaction, which does not encapsulate a handle, but manipulates the state of an object associated with a handle.

9.2.1. CSmartCardContext

Manages automatic context establishment and release and provides utility functions.

9.2.1.1. Local Type Definitions

An alias tstring is defined for std::basic_string<TCHAR>. Similar to TCHAR, for non-Unicode builds, tstring will map to std::string, for Unicode builds, it will map to std::wstring.

- `typedef std::basic_string<TCHAR> tstring;`

9.2.1.2. Constructor

- `explicit CSmartCardContext(const DWORD dwScope = SCARD_SCOPE_USER);`

  The constructor establishes the context by calling SCardEstablishContext(). The default scope is SCARD_SCOPE_USER, which can be overridden.

  The destructor will release the context by calling SCardReleaseContext().

9.2.1.3. Methods

- `SCARDCONTEXT GetHandle()`

  The method GetHandle() obtains the underlying handle and is used to directly call API Smart Card

functions which require a context handle.

- `std::list<tstring> EnumerateReaders() const`

  EnumerateReaders() returns a list of readers known to the system.

- `DWORD GetReaderStatus(const tstring &reader);`

  Obtains the current status for the specified reader. The result is a bitmask of `SCARD_STATE_XXX` constants, e.g. `SCARD_STATE_PRESENT` indicates that there is a card in the reader.

  The method calls SCardGetStatusChange() internally and returns the member dwEventState of the SCARD_READERSTATE struct.

- `vector<unsigned char> GetATRString(const tstring &reader);`

  Obtains the ATR string (Answer to Reset) generated for the tag in field (if any). Returns an empty vector if no tag is in the field.

9.2.2. CSmartCard

9.2.2.1. Local Type Definitions

The type tstring is defined locally (see CSmartCardContext):

- `typedef std::basic_string<TCHAR> tstring;`

9.2.2.2. Constructor

- ```
  CSmartCard(CSmartCardContext &context, const tstring &strReader,
             const DWORD dwShareMode = SCARD_SHARE_SHARED,
             const DWORD dwPrefProtocol = SCARD_PROTOCOL_T0 | SCARD_PROTOCOL_T1,
             const DWORD dwDisposition = SCARD_LEAVE_CARD);
  ```

Constructs a CSmartCard instance which forms the logical connection to a Smart Card/RFID Tag in the specified reader.

Please note that the connection can only be established once the tag is in the field of the reader (analogous to a card being inserted into the reader). The provided context must be valid during the overall lifetime of the class instance, i.e. it must not be deleted/destroyed before the instance of CSmartCard is destroyed/deleted.

Internally calls SCardConnect(). The destructor will call SCardDisconnect() with the parameter dwDisposition to release the connection.

The default parameters for dwShareMode, dwPrefProtocol and dwDisposition can be overridden, if necessary.
  ```

9.2.2.3. Methods

- `SCARDHANDLE GetHandle();`

  Obtains a handle to be used directly with the Smart Card API functions.

- `DWORD GetProtocol();`

  Obtains the negotiated protocol.

- ```
  void ExchangeAPDU(const std::vector<unsigned char> &apduOut,
            std::vector<unsigned char> &apduIn);
  ```

  Exchange APDUs with the tag. The APDU to be sent and received is passed as a vector. The APDU to be received (adpuIn) shall have a capacity matching the expected response, i.e. by calling apduIn.reserve(). If no memory was reserved, the maximum response APDU size (for a short APDU) will be allocated.

- ```
  size_t ExchangeAPDU(const unsigned char *const pbOut, const size_t cbOut,
            unsigned char* const pbIn, const size_t nCapacityIn);
  ```

  Exchange APDUs with the tag. The APDU to be sent and received is passed as void pointers together with the corresponding size parameters. The number of bytes actually received is returned.

- `void BeginTransaction();`

  Begin a transaction. Block other application from accessing the tag simultaneously, e.g. when using authentication functions, which require a strict order. The use of the CSmartCardTransaction class is highly recommended.

- `void EndTransaction(const DWORD dwDisposition = SCARD_LEAVE_CARD);`

  End a previously started transaction.

9.2.3. CSmartCardTransaction

Start a transaction, which spans the current scope. Calls CSmartCard::BeginTransaction() upon construction and CSmartCard::EndTransaction upon destruction.

This class provides a convenient and exception-safe way to handle transactions.

9.2.4. CSmartCardStatusReceiver

Provides a base class type for a status event receiver. An application must derive from this class to receive events in conjunction with CSmartCardStatusListener.

9.2.4.1. Local Type Definitions

The type tstring is defined locally (see CSmartCardContext):

- `typedef std::basic_string<TCHAR> tstring;`

9.2.4.2. Constructor

No constructor is explicitly declared. A default constructor will be generated.

9.2.4.3. Destructor

A virtual destructor is provided, in case an application might need it.

9.2.4.4. Methods

- `virtual void OnReaderAdded(const tstring &strReader);`

  This method will be called when a new reader is added to the system.

- `virtual void OnReaderRemoved(const tstring &strReader);`

  This method will be called when a reader is removed from the system.

- ```
  virtual void OnReaderEvent(const tstring &strReader,
            const DWORD dwCurrentState, const DWORD dwChanges,
            const std::vector<unsigned char>& vbATR);
  ```

  This method will be called for events on a specific card/tag. dwCurrentState indicates the current state and consists of SCARD_STATE_XXX flags. dwChanges encodes the same flags, but only indicates changes compared to the previous call. The ATR string (if available) is passed as a vector<unsigned char>.

  Please note that if a tag is lost and found again between two calls, this does not result in the SCARD_STATE_PRESENT flag being set in the dwChanges parameter. This change will only occur if the function got called at least once with SCARD_STATE_PRESENT cleared and again with SCARD_STATE_PRESENT.

9.2.5. CSmartCardStatusListener

Provides an easy way to build an event-driven application which automatically reacts to tag/card events, e.g. availability/non-availability of tags (inserted/ejected cards), and the addition and removal of reader devices.
  ```

9.2.5.1. Constructor

- `CSmartCardStatusListener(CSmartCardContext &context);`

  Constructs the instance in the given context. Please note that the context must be valid during the overall lifetime of this instance.

9.2.5.2. Methods

- `bool GetStatusChange(CSmartCardStatusReceiver &notify, DWORD dwTimepout = INFINITE);`

Get status changes. Waits until a status change has occurred or the specified timeout has elapsed.

Detected Events will be passed to the relevant methods of the specified class instance (derived from CSmartCardStatusReceiver).

9.2.6. CSmartCardExceptionHelper

The API uses exceptions to propagate unexpected error conditions, e.g. transmission problems.

It is left at the user's discretion, which exception class should be thrown. Two default implementations are provided. One class is derived from the C++ standard library class std::exception, whereas the other one is derived from the MFC CException class and can be used in MFC programs.

Please note that MFC exceptions must be caught by pointer, whereas ordinary C++-exceptions are usually caught by (const-) reference (or less-optimal by value).

CSmartCardExceptionHelper is a helper class, which provides static methods to throw an exception as well as methods to translate error codes returned from the Smart Card API.

9.2.6.1. Local Definitions

A enum is defined, enumerating the most common errors. Translation from error codes returned by the Smart Card API to this enum is done internally by the (private) static method `MapErrorCause()`.

- ```
  enum {
      errorUnknown,            // Unknown error, no mapping exists
      errorNoCard,             // No card in reader
      errorTransmission,       // Transmission failed
      errorInvalidParameter,   // Invalid Parameter
      errorUnknownReader,      // Reader not known
      errorTimeout,            // A timeout occurred
      errorSharing,            // A sharing violation occurred
      errorProtocolMismatch,   // Protocol mismatch
      errorUnavailable         // Reader or card/tag not available
  };
  ```

A sharing violation occurs if two applications try to open the same reader with incompatible settings, e.g. another application opened the reader in exclusive mode.
  ```

9.2.6.2. Methods

- `static const char * GetErrorCauseStr(const unsigned int nErrorCause);`

  Translates the enumerated error codes (c.f. 9.2.6.1) to a user-readable string.

- `static unsigned int MapErrorCause(const long lRet);`

  Translates the most common error codes returned by the Smart Card API to the enumerated error codes (c.f. 9.2.6.1)

- `static void Throw(long lRet, const std::string &strOp = "");`

  This function is invoked internally to throw an exception. `strOp` determines the operation (e.g. API call) which failed. `lRet` specifies the return value of the API function. Its value is internally translated to one of the enumerated error codes. This information is passed to the function ThrowImpl().

  ```
  static void ThrowImpl(const unsigned int nErrorCause, long lRet,
          const std::string &strOp);
  ```

  User-/application-defined function, which actually throws the exception and eventually determines the type of exception to throw.

  Two default implementations are provided (c.f. 9.2.7).

9.2.7. Pre-defined Exception Classes

SCardException.h and SCardException.cpp contain two pre-defined exception classes and the corresponding implementation of CSmartCardExceptionHelper::ThrowImpl().

Per default, the class CSmartCardExceptionStd will be declared and defined, which derives from std::exception. CSmartCardExceptionHelper::ThrowImpl() throws the exception by value.

By defining the pre-processor macro `SCARD_USE_MFC_EXCEPTIONS`, the alternative class CSmartCardExceptionMFC is declared and defined. The method CSmartCardExceptionHelper:: ThrowImpl() will throw allocate the exception on the heap and construct it by calling the new operator. A pointer to the exception will be thrown.

9.2.7.1. Common Data Fields

Both predefined exception classes contain the following public const data fields:

- `const unsigned int m_nErrorCause;`

  Contains the cause of the error, encoded by using the enum defined in CSmartCardExceptionHelper.

- `const long m_lReturn;`

  Contains the unmodified return value of the API function that failed.

- `const std::string m_strOperation;`

  Contains the name of the operation (e.g. API function) which failed.

9.2.7.2. Virtual Methods Overrides

The implementation based on the C++ class std::exception overrides the what() method:

- `virtual const char *what() const;`

The method returns a plain “CSmartCardException”, without the “std” prefix.

The MFC-based implementation overrides the GetErrorMessage() method:

```
virtual BOOL GetErrorMessage(LPTSTR lpszError, UINT nMaxError,
         PUNIT pnHelpContext = NULL);
```

The method returns a formatted error message, including the operation which failed, the API error code and the translated, human-readable error cause.

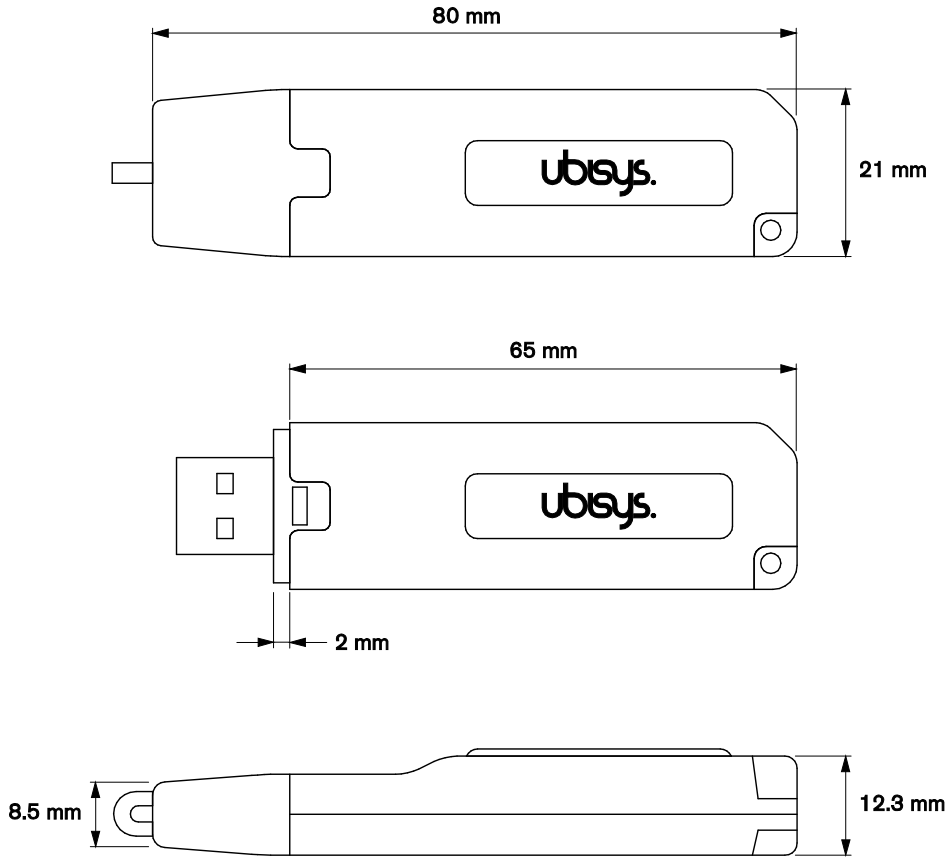

Figure 7: USB Stick with on-board PCB antenna

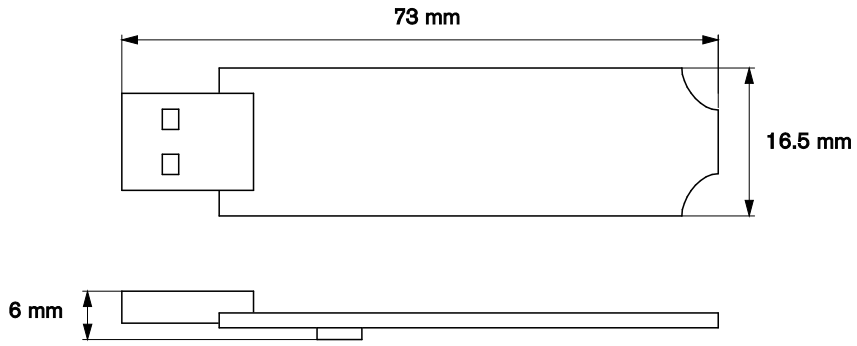

Figure 8: USB Stick with on-board PCB antenna (OEM, without case)

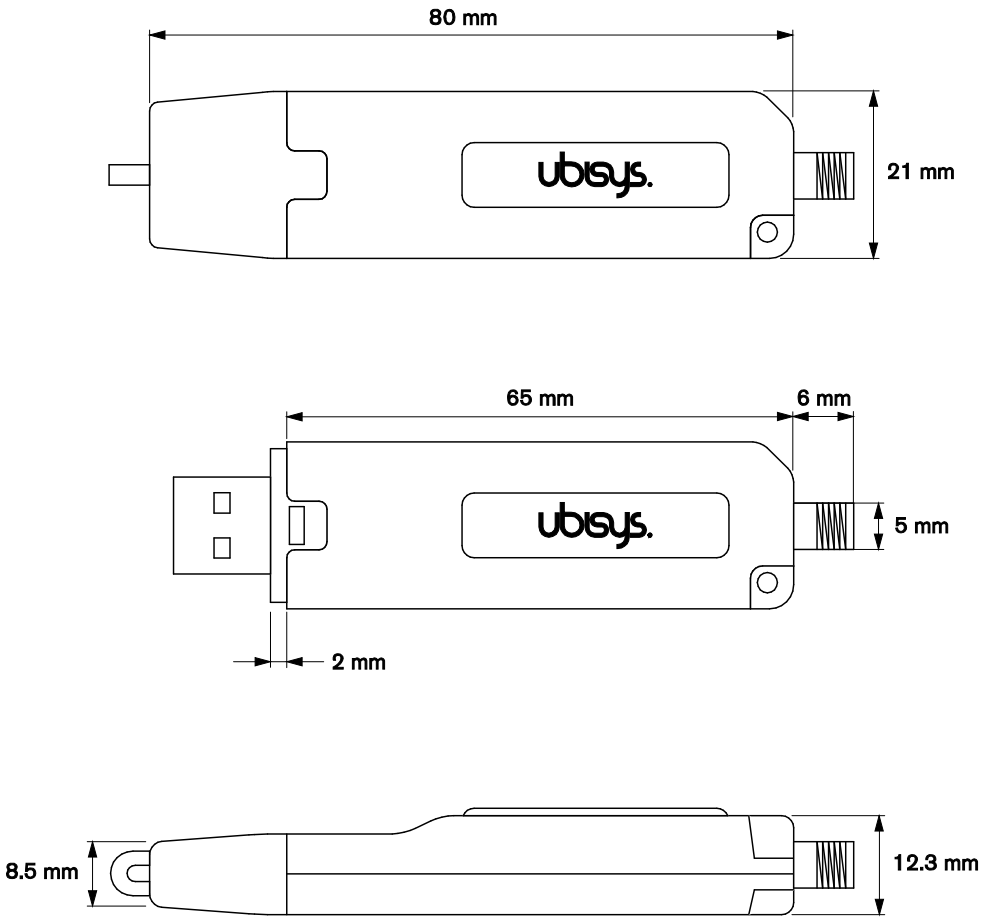

Figure 9: USB Stick with SMA Connector for External Antenna

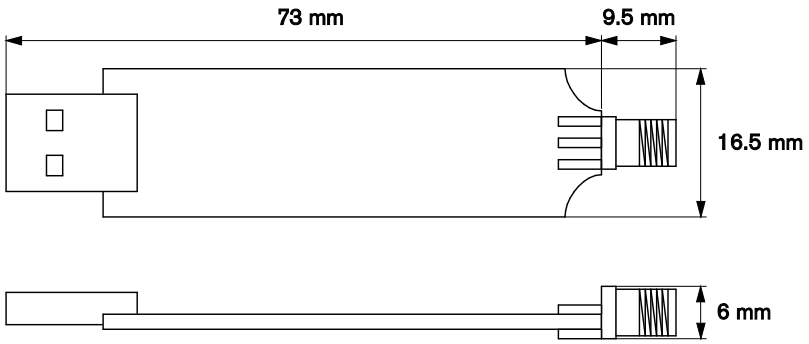

Figure 10: USB Stick with SMA Connector (OEM, without case)

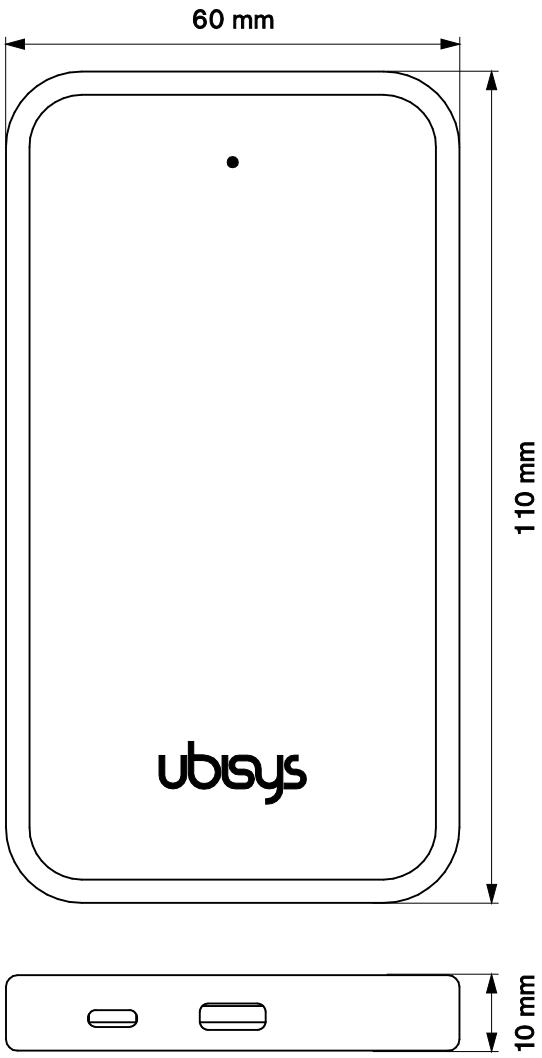

Figure 11: Desktop Reader Pro Case (USB Micro-B Socket)

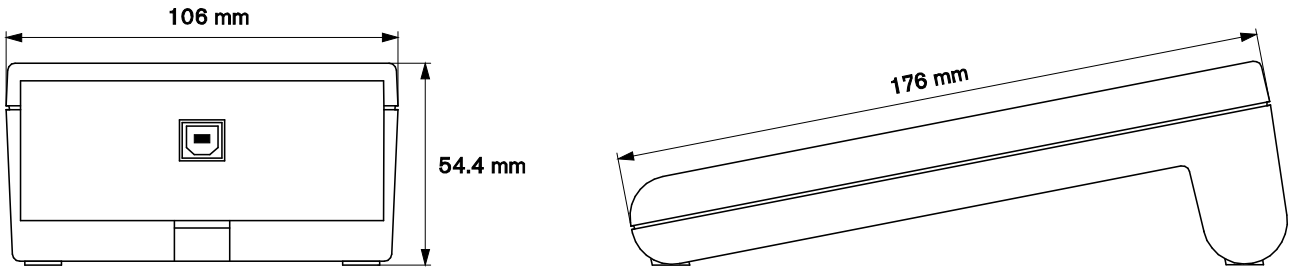

Figure 12: Desktop Reader Case (USB B Socket)

The following tables list the reader variants available. Use the specified order codes for your orders. Please contact ubisys support if you require any customization.

11.1. USB Stick

Antenna	Case	Firmware variant	Product Number	Order Code
On-board PCB antenna	Black	CDC	U0201-010120-01	**7030**
		HID	U0201-010120-02	**7047**
		CCID	U0201-010120-03	**7054**
	Grey	CDC	U0201-010220-01	**7061**
		HID	U0201-010220-02	**7078**
		CCID	U0201-010220-03	**7085**
	transparent	CDC	U0201-010320-01	**7092**
		HID	U0201-010320-02	**7108**
		CCID	U0201-010320-03	**7115**
	OEM (without case)	CDC	U0201-010020-01	**7009**
		HID	U0201-010020-02	**7016**
		CCID	U0201-010020-03	**7023**
SMA-Connector	Black	CDC	U0201-020120-01	**7153**
		HID	U0201-020120-02	**7160**
		CCID	U0201-020120-03	**7177**
	Grey	CDC	U0201-020220-01	**7184**
		HID	U0201-020220-02	**7191**
		CCID	U0201-020220-03	**7207**
	transparent	CDC	U0201-020320-01	**7214**
		HID	U0201-020320-02	**7221**
		CCID	U0201-020320-03	**7238**
	OEM (without case)	CDC	U0201-020020-01	**7122**
		HID	U0201-020020-02	**7139**
		CCID	U0201-020020-03	**7146**

11.2. Desktop Pro

Case	Firmware variant	Product Number	Order Code
Transparent & Black	CDC	U0209-010110-01	**7405**
	HID	U0209-010110-02	**7412**
	CCID	U0209-010110-03	**7429**

11.3. Desktop

Case	Firmware variant	Product Number	Order Code
Light Grey (RAL 7035)	CDC	U0202-010210-01	**7245**
	HID	U0202-010210-02	**7252**
	CCID	U0202-010210-03	**7269**

12. Declaration of Conformity

CE

We – ubisys technologies GmbH, Am Wehrhahn 45, 40211 Düsseldorf, Germany – declare under our sole responsibility that all orderable RFID products with product numbers and order codes as detailed in section 11 under the trade name “ubisys” to which this declaration relates are in conformity with the following directives and standards:

Directive/Standard	Description/Scope
2014/53/EU	Radio Equipment Directive (RED)
2014/30/EU	Electromagnetic Compatibility Directive (EMC)
2014/35/EU	Low Voltage Directive (LVD)
2012/19/EG	Waste Electrical and Electronic Equipment Directive (WEEE)
2011/65/EU	Restriction of Hazardous Substances Directive (RoHS)
EN 300 330	Spectrum
EN 301 489	EMC
EN 60950	Safety
EN 50364	Exposition
ISO 14443	RFID Proximity Cards
ISO 15693	RFID Vicinity Cards
ISO 18000-3	RFID Item Management

Düsseldorf, Germany	April 30, 2016
Place of issue	**Date of issue**
Dr.-Ing. Arasch Honarbacht	Managing Director, Head of Research & Development
Full name of Authorized Signatory	**Title of Authorized Signatory**
Arasch Honarbacht	ubisys. UBISYS TECHNOLOGIES GMBH HARDWARE UND SOFTWARE DESIGN ENGINEERING UND CONSULTING AM WEHRHAHN 45 40211 DÜSSELDORF info@ubisys.de www.ubisys.de
Signature	**Seal**

13. Document revision history

Revision	Date	Remarks
1.0	07/01/2009	Initial Version (CDC and HID firmware)
2.0	26/04/2010	Added CCID firmware documentation
2.1	18/04/2011	Updated ordering information. Initial public release
2.2	30/09/2011	CDC Firmware section contained outdated information on availability of 64-bit Windows drivers
2.3	25/10/2011	Updated list of supported tags
2.4	02/12/2011	Updated feature overview
2.5	15/12/2011	Updated CCID Get Data request documentation
2.6	20/01/2012	Corrected Supported TAGs section
2.7	23/03/2012	Updated PC/SC Lite & libccid information
2.8	10/04/2012	Added LED control command information. Minor editorial changes, corrected documentation for write single block command (CDC/ACM)
3.0	06/07/2012	Added Desktop Reader Pro dimensions and ordering information
3.1	19/07/2012	Added buzzer control command information
3.2	28/08/2012	RFID Control Center 1.4 supports firmware updates on 32- and 64-bit Windows platforms. Integrated Declaration of Conformity for European Union into manual
3.3	12/09/2012	Corrected compatibility matrix for Legic transponders
3.4	02/07/2014	Updated tag compatibility matrix. Added lock block command documentation. Fixed an error in the documentation for transparent APDU exchange with ISO 15693 tags using the firmware CCID (the P1 parameter was documented incorrectly).
3.5	03/21/2016	Updated declaration of conformity
3.6	04/30/2016	Updated declaration of conformity

14. Contact

UBISYS TECHNOLOGIES GMBH
HARDWARE AND SOFTWARE DESIGN
ENGINEERING AND CONSULTING

AM WEHRHAHN 45
40211 DÜSSELDORF
GERMANY

T: +49 (211) 54 21 55 - 00
F: +49 (211) 54 21 55 - 99

www.ubisys.de
info@ubisys.de
support@ubisys.de