**Interoperability Specification for ICCs and Personal Computer Systems**

*Part 10 IFDs with Secure PIN Entry Capabilities*

Apple Computer, Inc.

Gemalto, Inc.

Infineon Technologies AG

Ingenico SA

KOBIL

Microsoft Corporation

HID Global

Philips Semiconductors

SCM Microsystems

Toshiba Corporation

Vasco Data Security

**DRAFT Revision 2.02.10**

**October 2016**

**Copyright © 1996–2016 Apple, Gemalto, Hewlett-Packard, IBM, Infineon, Ingenico, Microsoft,   
HID Global, NXP Semiconductors, Oracle, Siemens, Toshiba, Vasco.  
All rights reserved.**

**INTELLECTUAL PROPERTY DISCLAIMER**

**THIS SPECIFICATION IS PROVIDED “AS IS” WITH NO WARRANTIES WHATSOEVER INCLUDING ANY WARRANTY OF MERCHANTABILITY, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION, OR SAMPLE.**

**NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED OR INTENDED HEREBY.**

**GEMalto, Hewlett-Packard, HID GLOBAL, IBM, Infineon, INGENICO, Microsoft, NXP Semicinductors, ORACLE CORPORATION, Siemens, and TOSHIBA DISCLAIM ALL LIABILITY, INCLUDING LIABILITY FOR INFRINGEMENT OF PROPRIETARY RIGHTS, RELATING TO IMPLEMENTATION OF INFORMATION IN THIS SPECIFICATION. GEMalto, Hewlett-Packard, HID Omnikey, IBM, Infineon, INGENICO, Microsoft, NXP Semicinductors, Siemens, SUN MICROSYSTEMS and TOSHIBA DO NOT WARRANT OR REPRESENT THAT SUCH IMPLEMENTATION(S) WILL NOT INFRINGE SUCH RIGHTS.**

Windows, Windows NT, Windows 2000, Windows 2003, Windows 2008, Windows 2012, Windows XP, Windows Vista, Windows 7, Windows 8 and Windows 10 are trademarks and Microsoft and Win32 are registered trademarks of Microsoft Corporation.  
PS/2 is a registered trademark of IBM Corp. JAVA is a registered trademark of Sun Microsystems, Inc. All other product names are trademarks, registered trademarks, or servicemarks of their respective owners.

**Revision History**

|  |  |  |
| --- | --- | --- |
| **Revision** | **Issue Date** | **Comments** |
| 2.00.00 | March 1st, 2005 | Spec 2.00 First Draft |
| 2.01.00 | March 8, 2005 | Spec 2.00 Reviewed Draft |
| 2.01.01 | March 24, 2005 | Spec 2.00 Reviewed Draft – minor revisions |
| 2.01.02 | April 19, 2005 | Minor edits |
| 2.01.03 | June 24, 2004 | Spec 2.01 Final Release |
| 2.01.04 | September 13, 2005 | Corrected data types in PIN structure |
| 2.01.05 | September 29, 2005 | Changed Schlumberger to Axalto |
| 2.01.06 | December 2, 2005 | Changes for GET\_KEY\_PRESSED |
| 2.02.00 | May 17, 2006 | New feature: applications can provide messages for secure pin entry  Cosmetic changes |
| 2.02.01 | November 7,2006 | IOCTL for writing to the display and retrieving key information added |
| 2.02.02 | March 20, 2007 | Microsoft: Added comments to address follow-ups from January 2007 Teleconference meeting. |
| 2.02.03 | October 2007 | Microsoft: updated based on comments from May 2007 meeting. |
| 2.02.04 | November 2008 | Gemalto: added warning regarding use of GEY\_KEY\_PRESSED and WRITE\_DISPLAY.  Added error code return when SET\_SPE\_MESSAGE exceeds storage capability.  Typo: changed all SPE\_SET\_MESSAGE to SET\_SPE\_MESSAGE for consistency  Added possible connection to the reader in 2.3 |
| 2.02.05 | December 2008 | Gemalto: changed ‘C’ structures to tables based of byte offsets; structure packing notes removed.  UTF-8 is now the only character set used.  “#define” feature removed from 2.3  Some typo and precision changes.  Structure and feature chapters created.  Rows and Columns (for WRITE\_DISPLAY and GEY\_KEY) start at index 0. |
| 2.02.06 | April 2009 | Gemalto: removed wLcdMaxCharacters and wLcdMaxLines fields from PIN\_PROPERTIES to maintain backward compatibility |
| 2.02.07 | March 2010 | Gemalto: Changed Axalto and Gemplus to Gemalto.  Gemalto: Added TLV properties feature.  Added bMinPINSize and bMaxPINSize tags.  Added sFirmwareID tag.  Added FEATURE\_CCID\_ESC\_COMMAND.  Added new features to all feature information example.  SCM Microsystems: Added Class 1 drivers to GET\_FEATURE\_REQUEST |
| 2.02.08 | April 2010 | Gemalto: Some typo/coloring fixed.  Added PPDUSupport as a TLV property. |
| 2.02.09 | November 2012 | Ingenico:  Added Extended APDU Support property as a new TLV property (dwMaxAPDUDataSize  ).  Added Reader Model property as two new TLV properties (wIdVendor and wIdProduct). |
| 2.02.10 | October 2016 | HID, Vasco: Added Floating PIN Frames.  Added Language Indicator.  Consolidated Amendment 1.1 and Supplement 1 into this document. |

Contents

1 Secure PIN Entry 1

1.1 System Architecture 1

2 Definition of Features and Control Codes 2

2.1 General Description 2

2.2 GET\_FEATURE\_REQUEST 2

2.3 Definition of Features 3

2.4 Function Call 5

2.5 Structure list 5

2.5.1 Type definitions 5

2.5.2 PIN\_VERIFY 6

2.5.3 PIN\_MODIFY 12

2.5.3.1 PIN\_MODIFY (Classic) 12

2.5.3.2 PIN\_MODIFY (Advanced) 17

2.5.4 MCT\_UNIVERSAL 23

2.5.5 PIN\_PROPERTIES 24

2.5.6 DISPLAY\_PROPERTIES 24

2.5.7 SET\_SPE\_MESSAGE 25

2.5.8 PIN\_VERIFY\_APP\_ID 26

2.5.9 PIN\_MODIFY\_APP\_ID 27

2.5.10 WRITE\_DISPLAY 28

2.5.11 GET\_KEY 28

2.5.12 EXECUTE\_PACE 29

2.6 Feature list 31

2.6.1 FEATURE\_VERIFY\_PIN\_START / FEATURE\_MODIFY\_PIN\_START 31

2.6.2 FEATURE\_GET\_KEY\_PRESSED 32

2.6.3 FEATURE\_VERIFY\_PIN\_FINISH / FEATURE\_MODIFY\_PIN\_FINISH 33

2.6.4 FEATURE\_VERIFY\_PIN\_DIRECT / FEATURE\_MODIFY\_PIN\_DIRECT 34

2.6.5 FEATURE\_ABORT 35

2.6.6 FEATURE\_MCT\_READER\_DIRECT 36

2.6.7 FEATURE\_MCT\_UNIVERSAL 36

2.6.8 FEATURE\_IFD\_PIN\_PROPERTIES 37

2.6.9 FEATURE\_IFD\_DISPLAY\_PROPERTIES 37

2.6.10 FEATURE\_SET\_SPE\_MESSAGE 38

2.6.11 FEATURE\_VERIFY\_PIN\_DIRECT\_APP\_ID / FEATURE\_MODIFY\_PIN\_DIRECT\_APP\_ID 39

2.6.12 FEATURE\_WRITE\_DISPLAY 40

2.6.13 FEATURE\_GET\_KEY 41

2.6.14 FEATURE\_GET\_TLV\_PROPERTIES 42

2.6.15 FEATURE\_CCID\_ESC\_COMMAND 44

2.6.16 FEATURE\_EXECUTE\_PACE 45

3 Feature Readers 48

3.1 System Architecture 48

3.2 Definition of Features 49

3.2.1 General Description 49

3.2.2 Feature Execution 49

3.2.2.1 Feature Execution by Pseudo-APDU 50

3.2.3 Get List of Features (GET\_FEATURE\_REQUEST) 51

3.2.3.1 GET\_FEATURE\_REQUEST by Pseudo-APDU 51

3.3 Features 52

3.3.1 FEATURE\_VERIFY\_PIN\_START 52

3.3.2 FEATURE\_VERIFY\_PIN\_FINISH 52

3.3.3 FEATURE\_MODIFY\_PIN\_START 52

3.3.4 FEATURE\_MODIFY\_PIN\_FINISH 53

3.3.5 FEATURE\_GET\_KEY\_PRESSED 53

3.3.6 FEATURE\_VERIFY\_PIN\_DIRECT 53

3.3.7 FEATURE\_MODIFY\_PIN\_DIRECT 53

3.3.8 FEATURE\_MCT\_READER\_DIRECT 54

3.3.9 FEATURE\_MCT\_UNIVERSAL 54

3.3.10 FEATURE\_IFD\_PIN\_PROPERTIES 54

3.3.11 FEATURE\_ABORT 55

3.3.12 FEATURE\_SET\_SPE\_MESSAGE 55

3.3.13 FEATURE\_VERIFY\_PIN\_DIRECT\_APP\_ID 55

3.3.14 FEATURE\_MODIFY\_PIN\_DIRECT\_APP\_ID 56

3.3.15 FEATURE\_WRITE\_DISPLAY 56

3.3.16 FEATURE\_GET\_KEY 56

3.3.17 FEATURE\_IFD\_DISPLAY\_PROPERTIES 56

3.3.18 FEATURE\_GET\_TLV\_PROPERTIES 57

3.3.19 FEATURE\_CCID\_ESC\_COMMAND 57

3.3.20 FEATURE\_EXECUTE\_PACE 57

Abbreviations 58

4 References 59

# Secure PIN Entry

## System Architecture

This section deals with secure PIN entry for class 2/3 readers and their integration into the PC/SC architecture.



Figure 1- General Architecture

Definition of Features and Control Codes

## General Description

An application queries the IFD handler via a special control code, whose features are supported by the IFD.

In the response, the application receives the control codes for all supported features. This mechanism enables different vendors/manufacturers of IFD handlers to define their own control codes.

The control codes, the features and the corresponding parameters are defined as follows.

## GET\_FEATURE\_REQUEST

The corresponding control code is 3400 decimal.

It is mandatory for class 2 drivers to support this control code.

**Out:**

The driver shall return Device\_Success and a TLV oriented structure as follows:

|  |  |  |
| --- | --- | --- |
| Field | Size (bytes) | Comment |
| Tag | 1 | See section 2.3 |
| Length | 1 | Must be set to 4 |
| Value | 4 | Control Code for supported feature |

The control code is returned in big endian format and must be used for the Control function of the Resource Manager in unmodified form.

Class 1 drivers should return Device\_Success and no TLV structures.

Implementation Note for Windows:

Device\_Success is implemented as SCARD\_S\_SUCCESS

Under Windows the control code must be defined as follows:

#define CM\_IOCTL\_GET\_FEATURE\_REQUEST SCARD\_CTL\_CODE(3400)

## Definition of Features

The following features are currently defined:

|  |  |
| --- | --- |
| FEATURE\_VERIFY\_PIN\_START | 0x01 |
| FEATURE\_VERIFY\_PIN\_FINISH | 0x02 |
| FEATURE\_MODIFY\_PIN\_START | 0x03 |
| FEATURE\_MODIFY\_PIN\_FINISH | 0x04 |
| FEATURE\_GET\_KEY\_PRESSED | 0x05 |
| FEATURE\_VERIFY\_PIN\_DIRECT | 0x06 |
| FEATURE\_MODIFY\_PIN\_DIRECT | 0x07 |
| FEATURE\_MCT\_READER\_DIRECT | 0x08 |
| FEATURE\_MCT\_UNIVERSAL | 0x09 |
| FEATURE\_IFD\_PIN\_PROPERTIES | 0x0A |
| FEATURE\_ABORT | 0x0B |
| FEATURE\_SET\_SPE\_MESSAGE | 0x0C |
| FEATURE\_VERIFY\_PIN\_DIRECT\_APP\_ID | 0x0D |
| FEATURE\_MODIFY\_PIN\_DIRECT\_APP\_ID | 0x0E |
| FEATURE\_WRITE\_DISPLAY | 0x0F |
| FEATURE\_GET\_KEY | 0x10 |
| FEATURE\_IFD\_DISPLAY\_PROPERTIES | 0x11 |
| FEATURE\_GET\_TLV\_PROPERTIES | 0x12 |
| FEATURE\_CCID\_ESC\_COMMAND | 0x13 |
| allocated, PC/SC defined usage | 0x14 – 0x1F |
| FEATURE\_EXECUTE\_PACE | 0x20 |
| allocated, PC/SC defined usage | 0x21 – 0x7F |
| non-allocated tags, for vendor specific use | 0x80 – 0xFE |
| future use | 0xFF |

e.g. an IFD handler which supports all features will return:

01 04 XX XX XX XX 02 04 XX XX XX XX 03 04 XX XX XX XX 04 04 XX XX XX XX 05 04 XX XX XX XX 06 04 XX XX XX XX 07 04 XX XX XX XX 08 04 XX XX XX XX 09 04 XX XX XX XX 0A 04 XX XX XX XX 0B 04 XX XX XX XX 0C 04 XX XX XX XX 0D 04 XX XX XX XX 0E 04 XX XX XX XX 0F 04 XX XX XX XX 10 04 XX XX XX XX 11 04 XX XX XX XX 12 04 XX XX XX XX 13 04 XX XX XX XX 20 04 XX XX XX XX

e.g. an IFD handler which supports none of the features will return no TLV structures. The Control function OutBufferLength value will be set to zero.

All control codes are sent to the IFD handler by the Control function of the Resource Manager. Before any control code can be used, a connection to the smart card or the reader is required. This can be achieved by the Connect function.

## Function Call

RESPONSECODE Control (

IN DWORD ControlCode,

IN BYTE[] InBuffer,

IN OUT BYTE[] Out Buffer,

OUT DWORD OutBufferLength

)

**Implementation Note for Windows**

Under Windows the following function must be used:

LONG SCardControl (

SCARDHANDLE hCard,

DWORD dwControlCode,

LPCVOID lpInBuffer,

DWORD nInBufferSize,

LPVOID lpOutBuffer,

DWORD nOutBufferSize,

LPDWORD lpBytesReturned

)

The total length of the TLV structure can be retrieved from the lpBytesReturned parameter of the SCardControl function.

When no TLV structures are present the lpBytesReturned value will be set to zero.

## Structure list

This chapter defines the structures exchanged to access the features listed in the next chapter.

### Type definitions

In this chapter, the data types are defined bellow:

|  |  |
| --- | --- |
| **Type** | **Size in bytes** |
| BYTE | 1 |
| USHORT | 2 |
| ULONG | 4 |

Byte ordering is decided by machine architecture.

Use of “[n]” after a type indicates an array of n elements of the given type.

If n is not specified, this means that the number of elements is specified by another field in the structure.

### PIN\_VERIFY

The PIN\_VERIFY structure is a set of instructions, such that the IFD can perform the following steps:

1. Display an *invitation text* on the display of the IFD
2. Let the user *input PIN digits* using the keypad of the IFD
3. Apply certain *formatting* on the entered PIN digits
4. Use *positioning* to move to certain location within APDU
5. Transfer the constructed APDU towards the smart card

Terminology:

“*PIN Length*” is the area where the number of entered PIN digits is stored.  
“*PIN Frame*” is the area where the entered PIN digits are stored.  
“abData” becomes the command APDU, to be sent to the smart card when all *input*, *formatting*, and *positioning* operations have completed:



Lc shall be re-calculated after the IFD device has constructed the APDU body.

**PIN\_VERIFY Structure**

| **Byte**  **Offset** | **Bit(s)** | **Field name** | **Type**  **(Value)** | **Description** |
| --- | --- | --- | --- | --- |
| 0 |  | bTimeOut | BYTE | Maximum time to finish the user input |
| 1..255 | Timeout in seconds |
| 0 | Default timeout |
| 1 |  | bTimeOut2 | BYTE | Maximum time after first keystroke |
| 0..255 | Timeout in seconds if “IFD distinguishes bTimeOut from bTimeOut2” (see 2.5.5)  Unused otherwise |
| 2 |  | bmFormatString |  |  |
| 7 | bPINFrameOffsetUnit | 1 BIT | Units (bit or byte) for the offset of the *PIN Frame* |
| 0 | Offset in bits |
| 1 | Offset in bytes |
| 6..3 | bPINFrameOffset | 4 BIT | *PIN Frame* offset value  Offset relative to *PIN Block* |
| 0..15 | Offset of the *PIN Frame* (offset in bits/bytes) |
| 2 | bPINFrameJustification | 1 BIT | Justification of PIN digits within the *PIN Frame* |
| 0 | Justify the PIN digits at left (fillers at right) |
| 1 | Justify the PIN digits at right (fillers at left) |
| 1..0 | bPINFrameCoding | 2 BIT | Coding of PIN digits in the *PIN Frame* |
| 0 | Binary coded (one byte per PIN digit) |
| 1 | BCD coded (half byte per PIN digit) |
| 2 | ASCII coded (one byte per PIN digit) |
| 3 | RFU |
| 3 |  | bmPINBlockString |  |  |
| 7..4 | bPINLengthSize | 4 BIT | Size (in bits) of *PIN Length* |
| 0 | The *PIN Length* is not used in the *PIN Block* |
| 1..15 | The size (in bits) of the *PIN Length* |
| 3..0 | bPINFrameSize | 4 BIT | Size (in bytes) of *PIN Frame* |
| 0 | Adaptive size (‘just fit’) of the *PIN Frame*  (1-byte placeholder is in the abData)  *This is an advanced feature, see ch 2.5.5* |
| 1..15 | size (*N* bytes) of the *PIN Frame*  (*N* placeholders in the abData) |
| 4 |  | bmPINLengthFormat |  |  |
| 7..5 |  | 3 BIT | RFU |
| 4 | bPINLengthOffsetUnit | 1 BIT | Units (bit or byte) for the offset of the *PIN Length* |
| 0 | Offset in bits |
| 1 | Offset in bytes |
| 3..0 | bPINLengthOffset | 4 BIT | *PIN Length* offset value  Offset relative to *PIN Block* |
| 0..15 | Offset of the *PIN Length* (offset in bits/bytes) |
| 5 |  | wPINMaxExtraDigit | USHORT |  |
| MSB | bPINMinimalDigits | 0..255 | Minimal number of PIN digits to be entered  (Most Significant Byte) |
| LSB | bPINMaximalDigits | 0..255 | Maximal number of PIN digits to be entered  (Least Significant Byte) |
| 7 |  | bEntryValidationCondition | BYTE | Collection of possible conditions when a PIN is considered to be complete |
| 7..3 |  | 5 BIT | RFU |
| 2 | ValidOnTimeout | 1 BIT | When ‘bTimeOut’ occurs: |
| 0 | The PIN entry is considered incomplete, verification shall be aborted |
| 1 | The PIN entry is considered complete, card verification shall be performed. |
| 1 | ValidationKeyPressed | 1 BIT | When the user hits the ‘Validate’ key  (and wPINMaxExtraDigit conditions are satisfied): |
| 0 | The ‘Validate’ key is ignored |
| 1 | The PIN entry is considered complete, card verification shall be performed. |
| 0 | MaxSizeReached | 1 BIT | When the bPINMaximalDigits are reached: |
| 0 | This condition is ignored |
| 1 | The PIN entry is considered complete, card verification shall be performed. |
| 8 |  | bNumberMessage | BYTE | Number of messages to display for PIN verification |
| 0 | No invitation message will be displayed |
| 1 | One invitation message shall be displayed |
| 2..254 | RFU |
| 255 | Use default CCID message |
| 9 |  | wLangId | USHORT | Language to be used for invitation message  (see http://www.usb.org/developers/docs/USB\_LANGIDs.pdf) |
| 11 |  | bMsgIndex | BYTE | Index of the invitation message |
| 0 | Invitation message nr 0 will be displayed  (e.g. “*Enter PIN:*”) |
| 1..255 | RFU |
| 12 |  | bTeoPrologue | BYTE[3] | T=1 I-block prologue field to use (fill with 00) |
| 15 |  | ulDataLength | ULONG | length of abData |
| 19 |  | abData | BYTE[] |  |
| BYTE[1..4] |  | *APDU header* of VERIFY command (e.g. 00 20 00 00) |
| BYTE[5] |  | (Placeholder) Length of the *APDU body* replaced by actual length of the final *APDU body* |
| BYTE[6..] |  | *APDU body*, holding *PIN Block* and fillers  *APDU body* expands with system fillers ‘FF’ if needed. |

Examples regarding “Display an *invitation text* on the display of the IFD”.

The *invitation text* is controlled by the fields bNumberMessage, wLangId and bMsgIndex.

|  |  |  |  |
| --- | --- | --- | --- |
| bNumber  Message | wLangId | bMsg  Index | Example *invitation text* on the display |
| 1 | 0x0409 | 1 | “Enter PIN:” |
| 1 | 0x080C | 1 | “Code PIN:” |
| 1 | 0x0407 | 1 | “PIN Code:” |

Examples regarding “user *input PIN digits* using the keypad of the IFD”.

The *input PIN digits* is controlled by the fields bTimeOut, bTimeOut2, bPINMinimalDigits, bPINMaximalDigits , ValidOnTimeout , ValidationKeyPressed and MaxSizeReached.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| bTimeOut | bTimeOut2 | bPINMinimal  Digits | bPINMaximal  Digits | ValidOn  Timeout | Validation  KeyPressed | MaxSize  Reached | Comment |
| 30 | 30 | 4 | 8 | 0 | 1 | 0 | Typical for EMV cards |
| 30 | 10 | 4 | 8 | 0 | 1 | 0 | After first digit PIN must be entered within 10 seconds |
| 30 | 30 | 6 | 6 | 0 | 1 | 0 | PIN requires exactly 6 digits |
| 30 | 30 | 6 | 6 | 0 | 0 | 1 | PIN complete when 6 digits entered |
| 30 | 30 | 4 | 8 | 1 | 1 | 0 | PIN complete on OK-key *or* after 30 seconds |

Examples regarding “*formatting* on the entered PIN digits”.

This *formatting* is controlled by the fields bPINFrameJustification, bPINFrameCoding, bPINLengthSize, and bPINFrameSize.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| bPINFrame  Justification | bPINFrame  Coding | bPINLength  Size (bits) | bPINFrame  Size (bytes) | Example PIN digits entered | PIN Length  (bits) | PIN Frame  (bytes)  (‘x’ are filler nibbles) |
| 0 | 1 | 4 | 7 | “12345” | b0101 | 12 34 5x xx xx xx xx |
| 0 | 2 | 4 | 7 | “12345” | b0101 | 31 32 33 34 35 xx xx |
| 1 | 1 | 4 | 7 | “12345” | b0101 | xx xx xx xx x1 23 45 |
| 1 | 2 | 4 | 7 | “12345” | b0101 | xx xx 31 32 33 34 35 |
| 0 | 1 | 8 | 8 | “12345” | b00000101 | 12 34 5x xx xx xx xx xx |
| 0 | 0 | 8 | 8 | “12345” | b00000101 | 01 02 03 04 05 xx xx xx |
| 0 | 1 | 8 | 0\* | “12345” | b00000101 | 12 34 5x |
| 1 | 1 | 8 | 0\* | “12345” | b00000101 | x1 23 45 |
| 0 | 2 | 8 | 0\* | “1234567” | b00000111 | 31 32 33 34 35 36 37 |

Remark: the items marked with (\*) are based on an IFD which supports ‘Adaptive PIN Frame size’.

Examples regarding “*positioning* to move to certain location within APDU”.

This *positioning* is controlled by the fields bPINFrameOffsetUnit, bPINFrameOffset, bPINLengthOffsetUnit and bPINLengthOffset.

|  |  |  |  |
| --- | --- | --- | --- |
| PIN Length | PIN Frame | Input areas | Resulting  APDU |
| Size  4 bit    Offset  4 bit | Size  7 byte    Offset  1 byte | PIN Length= b0101  PIN Frame= 12 34 5x xx xx xx xx  abData= 00 20 00 00 08  24 FF FF FF FF FF FF FF | 00 20 00 00 08  25 12 34 5F FF FF FF FF |
| Size  4 bit    Offset  4 bit | Size  7 byte    Offset  1 byte | PIN Length= b0101  PIN Frame= 31 32 33 34 35 xx xx  abData= 00 20 00 00 03  24 FF FF FF FF FF FF FF | 00 20 00 00 08  25 31 32 33 34 35 FF FF |
| Size  4 bit    Offset  4 bit | Size  7 byte    Offset  8 bit | PIN Length= b0101  PIN Frame= xx xx xx xx x1 23 45  abData= 00 20 00 00 03  24 FF FF FF FF FF FF FF | 00 20 00 00 08  25 FF FF FF FF F1 23 45 |
| Size  8 bit    Offset  1 byte | Size  7 byte    Offset  2 byte | PIN Length= b00000101  PIN Frame= 12 34 5x xx xx xx xx  abData= 00 20 00 00 00  11 24 FF FF FF FF FF FF | 00 20 00 00 09  11 05 12 34 5F FF FF FF FF |
| Size  8 bit    Offset  0 byte | Size  0\* byte  Offset  1 byte | PIN Length= b00000101  PIN Frame= 12 34 5x  abData= 00 20 00 00 00  77 FF | 00 20 00 00 04  05 12 34 5F |
| Size  8 bit    Offset  1 byte | Size  0\* byte    Offset  0 byte | PIN Length= b00000101  PIN Frame= x1 23 45  abData= 00 20 00 00 00  DE 77 88 | 00 20 00 00 05  D1 23 45 05 88 |
| Size  8 bit    Offset  0 bit | Size  0\* byte    Offset  1 byte | PIN Length= b00000111  PIN Frame= 31 32 33 34 35 36 37  abData= 00 20 00 00 00 | 00 20 00 00 08  07 31 32 33 34 35 36 37 |
| Size  0 bit    Offset  0 bit | Size  0\* byte    Offset  0 byte | PIN Length= b00000111  PIN Frame= 31 32 33 34 35 36 37  abData= 00 20 00 00 00 | 00 20 00 00 07  31 32 33 34 35 36 37 |

Remark: the items marked with (\*) are based on an IFD which supports ‘Adaptive PIN Frame size’.

The PIN Verify command can use a structure:

typedef struct \_PIN\_VERIFY\_STRUCTURE

{

BYTE bTimeOut; // timeout in seconds (00 means use default timeout)

BYTE bTimeOut2; // timeout in seconds after first keystroke

BYTE bmFormatString; // formatting options within PINBlock

BYTE bmPINBlockString; // define PINBlock

BYTE bmPINLengthFormat; // PINLength: number of PIN characters entered

USHORT wPINMaxExtraDigit; // XXYY, where XX is minimum PINLength in digits,

// YY is maximum

BYTE bEntryValidationCondition; // Conditions under which PIN entry should be

// considered complete

BYTE bNumberMessage; // Number of messages to use for PIN verification

USHORT wLangId; // Language for messages

BYTE bMsgIndex; // Message index (should be 00)

BYTE bTeoPrologue[3]; // T=1 I-block prologue field to use (fill with 00)

ULONG ulDataLength // length (in bytes) of ‘abData’

BYTE abData[1]; // Data (APDU) to send to the ICC

} PIN\_VERIFY\_STRUCTURE, \*PPIN\_VERIFY\_STRUCTURE;

A typical example for commanding PIN verification on an EMV card can use the following:

bTimeOut=0x1E; // 30 seconds timeout

bTimeOut2=0x1E; // 30 seconds timeout

bmFormatString=0x89; // at offset of 1 byte a left-justified PIN in BCD

bmPINBlockString=0x47; // PIN Length is 4-bits, PIN Frame is max 7 bytes

bmPINLengthFormat=0x04; // PIN Length to be inserted at offset of 4 bits

wPINMaxExtraDigit=0x0408; // minimal 4 and maximal 8 PIN digits to be entered

bEntryValidationCondition=0x02; // start validation after user hits the OK key

bNumberMessage=0x01; // just 1 message index is defined

wLangId=0x0409; // the English language, as used in USB

bMsgIndex=0x00; // the first message of the CCID table is “ENTER PIN”

bTeoPrologue=’00 00 00’; // is required in case of a T=1 card

ulDataLength = 0x0000000D;

abData=’00 20 00 80 08 20 FF FF FF FF FF FF FF’; // CardHolderVerification

A typical example for commanding PIN verification on an IAS/ECC card can use the following (a PINPAD reader with ‘Adaptive PIN Frame size’ is required):

bTimeOut=0x1E; // 30 seconds timeout

bTimeOut2=0x1E; // 30 seconds timeout

bmFormatString=0x82; // at offset 0 a PIN Frame in ASCII

bmPINBlockString=0x00; // PIN Length size is 0 bits, PIN Frame size is adaptive

bmPINLengthFormat=0x00; // -not used-

wPINMaxExtraDigit=0x0408; // minimal 4 and maximal 8 PIN digits to be entered

bEntryValidationCondition=0x02; // start validation after user hits the OK key

bNumberMessage=0x01; // just 1 message index is defined

wLangId=0x0409; // the English language, as used in USB

bMsgIndex=0x00; // the first message of the CCID table is “ENTER PIN”

bTeoPrologue=’00 00 00’; // is required in case of a T=1 card

ulDataLength = 0x0000000D;

abData=’00 20 00 00 00’; // CardHolderVerification

### PIN\_MODIFY

#### PIN\_MODIFY (Classic)

The PIN\_MODIFY structure is a set of instructions, such that the IFD can perform the following steps:

1. Display an *invitation text* on the display of the IFD
2. Let the user *input PIN digits* using the keypad of the IFD  
   (this can comprise old PIN, new PIN and/or confirmed new PIN)
3. Apply certain *formatting* on the entered PIN digits
4. Use *positioning* to move to certain location within APDU
5. Transfer the constructed APDU towards the smart card

Terminology:

“*PIN Length*” is the area where the number of entered PIN digits is stored.  
“*PIN Frame*” is the area where the entered PIN digits are stored.  
“abData” becomes the command APDU, to be sent to the smart card when all *input*, *formatting*, and *positioning* operations have completed.

“*PIN Block*” is the area of *PIN Length* and *PIN Frame*. This is used in the classic PIN modify command, where the “*PIN Block*” of the old PIN and the “*PIN Block*” of the new PIN have equal formats:



Lc shall be re-calculated after the IFD device has constructed the APDU body.

**PIN\_MODIFY Classic Structure**

| **Byte**  **Offset** | **Bit(s)** | **Field name** | **Type**  **(Value)** | **Description** |
| --- | --- | --- | --- | --- |
| 0 |  | bTimeOut | BYTE | Maximum time to finish the user input |
| 1..255 | Timeout in seconds |
| 0 | Default timeout |
| 1 |  | bTimeOut2 | BYTE | Maximum time after first keystroke |
| 0..255 | Timeout in seconds if “IFD distinguishes bTimeOut from bTimeOut2” (see 2.5.5)  Unused otherwise |
| 2 |  | bmFormatString |  |  |
| 7 | bPINFrameOffsetUnit | 1 BIT | Units (bit or byte) for the offset of the *PIN Frame* |
| 0 | Offset in bits |
| 1 | Offset in bytes |
| 6..3 | bPINFrameOffset | 4 BIT | *PIN Frame* offset value  Offset relative to *PIN Block* |
| 0..15 | Offset of the *PIN Frame* (offset in bits/bytes) |
| 2 | bPINFrameJustification | 1 BIT | Justification of PIN digits within the *PIN Frame* |
| 0 | Justify the PIN digits at left (fillers at right) |
| 1 | Justify the PIN digits at right (fillers at left) |
| 1..0 | bPINFrameCoding | 2 BIT | Coding of PIN digits in the *PIN Frame* |
| 0 | Binary coded (one byte per PIN digit) |
| 1 | BCD coded (half byte per PIN digit) |
| 2 | ASCII coded (one byte per PIN digit) |
| 3 | RFU |
| 3 |  | bmPINBlockString |  |  |
| 7..4 | bPINLengthSize | 4 BIT | Size (in bits) of *PIN Length* |
| 0 | The *PIN Length* is not used in the *PIN Block* |
| 1..15 | The size (in bits) of the *PIN Length* |
| 3..0 | bPINFrameSize | 4 BIT | Size (in bytes) of *PIN Frame* |
| 0 | Adaptive size (‘just fit’) of the *PIN Frame*  (1-byte placeholder is in the abData) |
| 1..15 | N- the fixed size (in bytes) of the *PIN Frame*  (N placeholders in the abData) |
| 4 |  | bmPINLengthFormat |  |  |
| 7..5 |  | 3 BIT | RFU |
| 4 | bPINLengthOffsetUnit | 1 BIT | Units (bit or byte) for the offset of the *PIN Length* |
| 0 | Offset in bits |
| 1 | Offset in bytes |
| 3..0 | bPINLengthOffset | 4 BIT | *PIN Length* offset value  Offset relative to *PIN Block* |
| 0..15 | Offset of the *PIN Length* (offset in bits/bytes) |
| 5 |  | bInsertionOffsetOld | BYTE | Insertion position offset in bytes for the current PIN |
| 0..255 | Offset of current *PIN block*, relative to *APDU body* |
| 6 |  | bInsertionOffsetNew | BYTE | Insertion position offset in bytes for the new PIN |
| 0..255 | Offset of new *PIN block*, relative to *APDU body* |
| 7 |  | wPINMaxExtraDigit | USHORT |  |
| MSB | bPINMinimalDigits | 0..255 | Minimal number of PIN digits to be entered  (Most Significant Byte) |
| LSB | bPINMaximalDigits | 0..255 | Maximal number of PIN digits to be entered  (Least Significant Byte) |
| 9 |  | bConfirmPin |  | Flags governing need for confirmation of new PIN |
| 7..3 |  | 5 BIT | RFU, shall be 0 |
| 2 | Advanced Modify | 1 BIT | Which Modify structure applies: Classic or Advanced |
| 0 | Classic Modify Structure (this chapter)  Default value. |
| 1 | Advanced Modify Structure (see chapter 2.5.3.2) |
| 1 | RequestCurrentPin | 1 BIT | Shall the current PIN be requested and positioned? |
| 0 | Do not use current PIN (‘old PIN’) |
| 1 | Request and position current PIN (‘old PIN’) |
| 0 | ConfirmNewPin | 1 BIT | Shall the new PIN be confirmed? |
| 0 | Do not request ‘Confirm new PIN’ |
| 1 | Request ‘Confirm new PIN’ and verify equality |
| 10 |  | bEntryValidationCondition | BYTE | Collection of possible conditions when a PIN is considered to be complete |
| 7..3 |  | 5 BIT | RFU |
| 2 | ValidOnTimeout | 1 BIT | When ‘bTimeOut’ occurs: |
| 0 | The PIN entry is considered incomplete, verification shall be aborted |
| 1 | The PIN entry is considered complete, card verification shall be performed. |
| 1 | ValidationKeyPressed | 1 BIT | When the user hits the ‘Validate’ key  (and wPINMaxExtraDigit conditions are satisfied): |
| 0 | The ‘Validate’ key is ignored |
| 1 | The PIN entry is considered complete, card verification shall be performed. |
| 0 | MaxSizeReached | 1 BIT | When the bPINMaximalDigits are reached: |
| 0 | This condition is ignored |
| 1 | The PIN entry is considered complete, card verification shall be performed. |
| 11 |  | bNumberMessage | BYTE | Number of messages to display for PIN verification |
| 0 | No invitation message will be displayed |
| 1 | One invitation message shall be displayed |
| 2 | Two invitation messages shall be displayed |
| 3 | Three invitation messages shall be displayed |
| 4..254 | RFU |
| 255 | Use default CCID message |
| 12 |  | wLangId | USHORT | Language to be used for invitation message  (see http://www.usb.org/developers/docs/USB\_LANGIDs.pdf) |
| 14 |  | bMsgIndex1 | BYTE | Index of the invitation message |
| 0 | Invitation message nr 0 shall be displayed  (e.g. “*Enter current PIN:*”) |
| 1 | Invitation message nr 1 shall be displayed  (e.g. “*Enter new PIN:*”) |
| 3..255 | RFU |
| 15 |  | bMsgIndex2 | BYTE | Index of the invitation message |
| 1 | Invitation message nr 1 will be displayed  (e.g. “*Enter new PIN:*”) |
| 2 | Invitation message nr 2 will be displayed  (e.g. “*Repeat new PIN:*”) |
| 3..255 | RFU |
| 16 |  | bMsgIndex3 | BYTE | Index of the invitation message |
| 2 | Invitation message nr 2 will be displayed  (e.g. “*Repeat new PIN:*”) |
| 3..255 | RFU |
| 17 |  | bTeoPrologue | BYTE[3] | T=1 I-block prologue field to use (fill with 00) |
| 20 |  | ulDataLength | ULONG | length of abData |
| 24 |  | abData | BYTE[] |  |
| BYTE[1..4] |  | *APDU header* of MODIFY command (e.g. 00 24 00 00) |
| BYTE[5] |  | (Placeholder) Length of the *APDU body* replaced by actual length of the final *APDU body* |
| BYTE[6..] |  | *APDU body*, holding 2 *PIN Block*s and fillers  *APDU body* expands with system fillers ‘FF’ if needed. |

Examples regarding “*positioning* to move to certain location within APDU”.

This *positioning* is controlled by the fields bPINFrame(Offset & unit), bPINLength(Offset & unit), bmPINBlockString (PINFrame Size & PINLength Size), bInsertionOffsetOld and bInsertionOffsetNew.

| PIN Length | PIN Frame | bInsertion  OffsetOld | bInsertion  OffsetNew | Input areas | Resulting  APDU |
| --- | --- | --- | --- | --- | --- |
| Size  4 bit    Offset  4 bit | Size  7 byte    Offset  1 byte | 0 | 8 | oldPIN Length= b0101  oldPIN Frame= 12 34 5x xx xx xx xx  newPIN Length= b0111  newPIN Frame= 12 34 56 7x xx xx xx  abData= 00 24 00 00 10  24 FF FF FF FF FF FF FF  24 FF FF FF FF FF FF FF | 00 24 00 00 10  25 12 34 5F FF FF FF FF  27 12 34 56 7F FF FF FF |
| Size  8 bit    Offset  0 bit | Size  7 byte    Offset  1 byte | 0 | 8 | oldPIN Length= b0101  oldPIN Frame= 12 34 5x xx xx xx xx  newPIN Length= b0111  newPIN Frame= 12 34 56 7x xx xx xx  abData= 00 24 00 00 00 | 00 24 00 00 10  05 12 34 5F FF FF FF FF  07 12 34 56 7F FF FF FF |
| Size  8 bit    Offset  0 bit | Size  0\* byte    Offset  1 byte | 0 | 2 | oldPIN Length= b0101  oldPIN Frame= 12 34 5x  newPIN Length= b0111  newPIN Frame= 12 34 56 7x  abData= 00 24 00 00 04  00 EE 00 EE | 00 24 00 00 09  05 12 34 5E  07 12 34 56 7E |

Remark: the items marked with (\*) are based on an IFD which supports ‘Adaptive PIN Frame size’.

The PIN Modify command can be based on a structure:

typedef struct \_PIN\_MODIFY\_STRUCTURE

{

BYTE bTimeOut; // timeout in seconds (00 means use default timeout)

BYTE bTimeOut2; // timeout in seconds after first keystroke

BYTE bmFormatString; // formatting options within PINBlock

BYTE bmPINBlockString; // define PINBlock

BYTE bmPINLengthFormat; // PINLength: number of PIN characters entered

BYTE bInsertionOffsetOld; // Insertion position offset in bytes for the current PIN

BYTE bInsertionOffsetNew; // Insertion position offset in bytes for the new PIN

USHORT wPINMaxExtraDigit; // XXYY, where XX is minimum PINLength in digits,

// YY is maximum

BYTE bConfirmPIN; // Flags governing need for confirmation of new PIN

BYTE bEntryValidationCondition; // Conditions under which PIN entry should be

// considered complete

BYTE bNumberMessage; // Number of messages to use for PIN verification

USHORT wLangId; // Language for messages

BYTE bMsgIndex1; // Index of 1st prompting message

BYTE bMsgIndex2; // Index of 2d prompting message (if appropriate)

BYTE bMsgIndex3; // Index of 3d prompting message (if appropriate)

BYTE bTeoPrologue[3]; // T=1 I-block prologue field to use (fill with 00)

ULONG ulDataLength // length (in bytes) of ‘abData’

BYTE abData[1]; // Data to send to the ICC (this is a placeholder)

} PIN\_MODIFY\_STRUCTURE, \*PPIN\_MODIFY\_STRUCTURE;

A typical example for commanding PIN modification (Classic) on an IAS/ECC card can use the following (a PINPAD reader with ‘Adaptive PIN Frame size’ is required):

bTimeOut=0x1E; // 30 seconds timeout

bTimeOut2=0x1E; // 30 seconds timeout

bmFormatString=0x82; // at offset of 0 byte a PIN in ASCII

bmPINBlockString=0x00; // PIN Length size is 0 bits, PIN Frame size is adaptive

bmPINLengthFormat=0x00; // -not used-

bInsertionOffsetOld=0x00; // current PIN block at adaptive offset 0

bInsertionOffsetNew=0x01; // new PIN block at adaptive offset 1

bConfirmPIN=0x03; // Request enter current PIN, Request Confirm new PIN

wPINMaxExtraDigit=0x0408; // minimal 4 and maximal 8 PIN digits to be entered

bEntryValidationCondition=0x02; // start validation after user hits the OK key

bNumberMessage=0x03; // 3 message indices are defined

wLangId=0x0409; // the English language, as used in USB

bMsgIndex1=0x00; // the first message of the CCID table is “ENTER PIN”

bMsgIndex2=0x01; // the second message of CCID is “ENTER NEW PIN”

bMsgIndex3=0x02; // the third message of CCID is “REPEAT NEW PIN”

bTeoPrologue=’00 00 00’; // is required in case of a T=1 card

ulDataLength = 0x0000000D;

abData=’00 24 00 80 00’; // Change Reference Data APDU

#### PIN\_MODIFY (Advanced)

The PIN\_MODIFY *advanced* structure is a set of instructions, such that the IFD can perform the following steps:

1. Display an *invitation text* on the display of the IFD
2. Let the user *input PIN digits* using the keypad of the IFD
3. Apply certain *formatting* on the entered PIN digits
4. Use *positioning* to move to certain location within APDU
5. Transfer the constructed APDU towards the smart card

The IFD supports PIN\_MODIFY *advanced* structure if the PIN\_PROPERTIES structure indicates this support (see ch 2.5.5)

Terminology:

“*PIN Length*” is the area where the number of entered PIN digits is stored.  
“*PIN Frame*” is the area where the entered PIN digits are stored.  
“abData” becomes the command APDU, to be sent to the smart card when all *input*, *formatting*, and *positioning* operations have completed: Lc shall be re-calculated after the IFD device has constructed the APDU body.



**PIN\_MODIFY *Advanced* Structure**

| **Byte**  **Offset** | **Bit(s)** | **Field name** | **Type**  **(Value)** | **Description** |
| --- | --- | --- | --- | --- |
| 0 |  | bTimeOut | BYTE | Maximum time to finish the user input |
| 1..255 | Timeout in seconds |
| 0 | Default timeout |
| 1 |  | bTimeOut2 | BYTE | Maximum time after first keystroke |
| 0..255 | Timeout in seconds if “IFD distinguishes bTimeOut from bTimeOut2” (see 2.5.5)  Unused otherwise |
| 2 |  | bmFormatString |  |  |
| 7 | bPINFrameOffsetUnit | 1 BIT | Units (bit or byte) for the offset of both NewPINFrame and OldPINFrame |
| 0 | Offset in bits |
| 1 | Offset in bytes |
| 6..3 | bOldPINFrameOffset | 4 BIT | PIN Frame offset value of the Old PIN  Offset relative to *APDUbody* |
| 0..15 | Offset of the OldPINFrame (offset in bits/bytes, see byte 2 bit 7) |
| 2 | bPINFrameJustification | 1 BIT | Justification of PIN digits within both NewPINFrame and OldPINFrame |
| 0 | Justify the PIN digits at left (fillers at right) |
| 1 | Justify the PIN digits at right (fillers at left) |
| 1..0 | bPINFrameCoding | 2 BIT | Coding of PIN digits in both NewPINFrame and OldPINFrame |
| 0 | Binary coded (one byte per PIN digit) |
| 1 | BCD coded (half byte per PIN digit) |
| 2 | ASCII coded (one byte per PIN digit) |
| 3 | RFU |
| 3 |  | bmPINBlockString |  |  |
| 7..4 | bPINLengthSize | 4 BIT | Size (in bits) of both NewPINLength and OldPINLength |
| 0 | The *PIN Length* is not used |
| 1..15 | The size (in bits) of both *PIN Length*’s |
| 3..0 | bPINFrameSize | 4 BIT | Size (in bytes) of both NewPINFrame and OldPINFrame |
| 0 | Adaptive size (‘just fit’) of both *PIN Frame*’s  (1-byte placeholder is in the abData) |
| 1..15 | N- the fixed size (in bytes) of both *PIN Frame*’s  (N placeholders in the abData) |
| 4 |  | bmPINLengthFormat |  |  |
| 7..5 |  | 3 BIT | RFU |
| 4 | bPINLengthOffsetUnit | 1 BIT | Units (bit or byte) for the offset of both NewPINLength and the OldPINLength |
| 0 | Offset in bits |
| 1 | Offset in bytes |
| 3..0 | bOldPINLengthOffset | 4 BIT | Old PIN Length offset value  Offset relative to *APDUbody* |
| 0..15 | Offset of the OldPINLength (offset in bits/bytes, see byte 4 bit 4) |
| 5 |  | bNewPINLengthOffset | BYTE | Offset of the NewPINLength (offset in bits/bytes, see byte 4 bit 4) |
| 0..255 | Offset of new *PIN Length*, relative to *APDU body* |
| 6 |  | bNewPINFrameOffset | BYTE | Offset of the NewPINFrame (offset in bits/bytes, see byte 2 bit 7) |
| 0..255 | Offset of new *PIN Frame*, relative to *APDU body* |
| 7 |  | wPINMaxExtraDigit | USHORT |  |
| MSB | bPINMinimalDigits | 0..255 | Minimal number of PIN digits to be entered  (Most Significant Byte) |
| LSB | bPINMaximalDigits | 0..255 | Maximal number of PIN digits to be entered  (Least Significant Byte) |
| 9 |  | bConfirmPin |  | Flags governing need for confirmation of new PIN |
| 7..3 |  | 5 BIT | RFU, shall be 0 |
| 2 | Advanced Modify | 1 BIT | Which Modify structure applies: Classic or Advanced |
| 0 | Classic Modify Structure (see chapter 2.5.3.1)  Default value. |
| 1 | Advanced Modify Structure (this chapter) |
| 1 | RequestCurrentPin | 1 BIT | Shall the current PIN be requested and positioned? |
| 0 | Do not use current PIN (‘old PIN’) |
| 1 | Request and position current PIN (‘old PIN’) |
| 0 | ConfirmNewPin | 1 BIT | Shall the new PIN be confirmed? |
| 0 | Do not request ‘Confirm new PIN’ |
| 1 | Request ‘Confirm new PIN’ and verify equality |
| 10 |  | bEntryValidationCondition | BYTE | Collection of possible conditions when a PIN is considered to be complete |
| 7..3 |  | 5 BIT | RFU |
| 2 | ValidOnTimeout | 1 BIT | When ‘bTimeOut’ occurs: |
| 0 | The PIN entry is considered incomplete, verification shall be aborted |
| 1 | The PIN entry is considered complete, card verification shall be performed. |
| 1 | ValidationKeyPressed | 1 BIT | When the user hits the ‘Validate’ key  (and wPINMaxExtraDigit conditions are satisfied): |
| 0 | The ‘Validate’ key is ignored |
| 1 | The PIN entry is considered complete, card verification shall be performed. |
| 0 | MaxSizeReached | 1 BIT | When the bPINMaximalDigits are reached: |
| 0 | This condition is ignored |
| 1 | The PIN entry is considered complete, card verification shall be performed. |
| 11 |  | bNumberMessage | BYTE | Number of messages to display for PIN verification |
| 0 | No invitation message will be displayed |
| 1 | One invitation message shall be displayed  (being at bMsgIndex1) |
| 2 | Two invitation messages shall be displayed  (being at bMsgIndex1 and bMsgIndex2) |
| 3 | Three invitation messages shall be displayed  (being at bMsgIndex1 and bMsgIndex2 and bMsgIndex3) |
| 4..254 | RFU |
| 255 | Use default CCID message |
| 12 |  | wLangId | USHORT | Language to be used for invitation message  (see http://www.usb.org/developers/docs/USB\_LANGIDs.pdf) |
| 14 |  | bMsgIndex1 | BYTE | Index of the invitation message |
| 0 | Invitation message nr 0 will be displayed  (e.g. “*Enter current PIN:*”) |
| 3..255 | RFU |
| 15 |  | bMsgIndex2 | BYTE | Index of the invitation message |
| 1 | Invitation message nr 1 will be displayed  (e.g. “*Enter new PIN:*”) |
| 3..255 | RFU |
| 16 |  | bMsgIndex3 | BYTE | Index of the invitation message |
| 2 | Invitation message nr 2 will be displayed  (e.g. “*Repeat new PIN:*”) |
| 3..255 | RFU |
| 17 |  | bTeoPrologue | BYTE[3] | T=1 I-block prologue field to use (fill with 00) |
| 20 |  | ulDataLength | ULONG | length of abData |
| 24 |  | abData | BYTE[] |  |
| BYTE[1..4] |  | *APDU header* of MODIFY command (e.g. 00 24 00 00) |
| BYTE[5] |  | (Placeholder) Length of the *APDU body* replaced by actual length of the final *APDU body* |
| BYTE[6..] |  | *APDU body*, holding OldPINLength, OldPINFrame, NewPINLength, NewPINFrame and fillers  *APDU body* expands with system fillers ‘FF’ if needed. |

Examples regarding “*positioning* to move to certain location within APDU”.

This *positioning* is controlled by the fields bmFormatString(PINFrame Offset\_unit & OldPINFrameOffset), bmPINBlockString (PINFrame Size & PINLength Size), bmPINLengthFormat (PINLength Offset\_unit & OldPINLengthOffset), bNewPINLengthOffset, and bNewPINFrameOffset.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| PIN Length | PIN Frame | bOldPINLength  Offset (0..15) | bNewPINLength  Offset (0..255) | bOldPINFrame  Offset (0..15) | bNewPINFrame  Offset (0..255) | Input area’s | Resulting  APDU |
| Size  4 bit | Size  7 byte | 4  bit | 68  bit | 1 byte | 9 byte | oldPIN Length= b0101  oldPIN Frame= 12 34 5x xx xx xx xx  newPIN Length= b0111  newPIN Frame= 12 34 56 7x xx xx xx  abData= 00 24 00 00 10  20 FF FF FF FF FF FF FF  20 FF FF FF FF FF FF FF | 00 24 00 00 10  25 12 34 5F FF FF FF FF  27 12 34 56 7F FF FF FF |
| Size  8 bit | Size  7 byte | 0 byte | 8 byte | 1 byte | 9 byte | oldPIN Length= b0101  oldPIN Frame= 12 34 5x xx xx xx xx  newPIN Length= b0111  newPIN Frame= 12 34 56 7x xx xx xx  abData= 00 24 00 00 00 | 00 24 00 00 10  05 12 34 5F FF FF FF FF  07 12 34 56 7F FF FF FF |
| Size  0 bit | Size  8 byte | 0 byte | 0 byte | 0 byte | 8 byte | oldPIN Length= empty  oldPIN Frame= 12 34 5x xx xx xx xx xx  newPIN Length= empty  newPIN Frame= 12 34 56 7x xx xx xx xx  abData= 00 24 00 00 00 | 00 24 00 00 10  12 34 5F FF FF FF FF FF  12 34 56 7F FF FF FF FF |
| Size  8 bit | Size  0\* byte | 0  byte | 1  byte | 2  byte | 3  byte | oldPIN Length= b00000101  oldPIN Frame= 12 34 5x  newPIN Length= b00000111  newPIN Frame= 12 34 56 7x  abData= 00 24 00 80 04  CC DD EE EE | 00 24 00 80 09  05 07  12 34 5E  12 34 56 7E |
| Size  0 bit | Size  0\* byte | 0  byte | 0  byte | 0  byte | 1  byte | oldPIN Length= b00000101  oldPIN Frame= 31 32 33 34 35  newPIN Length= b00000111  newPIN Frame= 31 32 33 34 35 36 37  abData= 00 24 00 80 00 | 00 24 00 80 0C  31 32 33 34 35  31 32 33 34 35 36 37 |

Remark: the items marked with (\*) are based on an IFD which supports ‘Adaptive PIN Frame size’.

The PIN Modify command can be based on a structure:

typedef struct \_PIN\_MODIFY\_STRUCTURE

{

BYTE bTimeOut; // timeout in seconds (00 means use default timeout)

BYTE bTimeOut2; // timeout in seconds after first keystroke

BYTE bmFormatString; // formatting options within PINBlock

BYTE bmPINBlockString; // define PINBlock

BYTE bmPINLengthFormat; // PINLength: number of PIN characters entered

BYTE bInsertionOffsetOld; // Insertion position offset in bytes for the current PIN

BYTE bInsertionOffsetNew; // Insertion position offset in bytes for the new PIN

USHORT wPINMaxExtraDigit; // XXYY, where XX is minimum PINLength in digits,

// YY is maximum

BYTE bConfirmPIN; // Flags governing need for confirmation of new PIN

BYTE bEntryValidationCondition; // Conditions under which PIN entry should be

// considered complete

BYTE bNumberMessage; // Number of messages to use for PIN verification

USHORT wLangId; // Language for messages

BYTE bMsgIndex1; // Index of 1st prompting message

BYTE bMsgIndex2; // Index of 2d prompting message (if appropriate)

BYTE bMsgIndex3; // Index of 3d prompting message if appropriate)

BYTE bTeoPrologue[3]; // T=1 I-block prologue field to use (fill with 00)

ULONG ulDataLength // length (in bytes) of ‘abData’

BYTE abData[1]; // Data to send to the ICC (this is a placeholder)

} PIN\_MODIFY\_STRUCTURE, \*PPIN\_MODIFY\_STRUCTURE;

A typical example for commanding PIN modification (Advanced) on an IAS/ECC card can use the following (a PINPAD reader with ‘Adaptive PIN Frame size’ is required):

bTimeOut=0x1E; // 30 seconds timeout

bTimeOut2=0x1E; // 30 seconds timeout

bmFormatString=0x82; // at offset of 0 byte the Old PIN in ASCII

bmPINBlockString=0x00; // PIN Length size is 0 bits, PIN Frame size is adaptive

bmPINLengthFormat=0x00; // -not used-

bInsertionOffsetOld=0x00; // current PIN block at adaptive offset 0

bInsertionOffsetNew=0x01; // new PIN block at adaptive offset 1

bConfirmPIN=0x07; // Request current PIN & Confirm new PIN, use Advanced

wPINMaxExtraDigit=0x0408; // minimal 4 and maximal 8 PIN digits to be entered

bEntryValidationCondition=0x02; // start validation after user hits the OK key

bNumberMessage=0x03; // 3 message indices are defined

wLangId=0x0409; // the English language, as used in USB

bMsgIndex1=0x00; // the first message of the CCID table is “ENTER PIN”

bMsgIndex2=0x01; // the second message of CCID is “ENTER NEW PIN”

bMsgIndex3=0x02; // the third message of CCID is “REPEAT NEW PIN”

bTeoPrologue='00 00 00'; // is required in case of a T=1 card

ulDataLength = 0x0000000D;

abData='00 24 00 80 00'; // Change Reference Data APDU







### MCT\_UNIVERSAL

|  |  |  |  |
| --- | --- | --- | --- |
| **Byte offset** | **Field** | **Type** | **Description** |
| 0 | SAD | BYTE | Source ADdress, see [2] |
| 1 | DAD | BYTE | Destination ADdress, see [2] |
| 2 | BufferLength | USHORT | Size in bytes of the following buffer |
| 4 | Buffer | BYTE[] | Buffer to send to the device |

### PIN\_PROPERTIES



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Byte offset** | **Bit(s)** | **Field** | **Type**  **(Value)** | **Description** |
| 0 |  | wLcdLayout | USHORT | display characteristics as defined in [4] |
| 2 |  | bEntryValidation  Condition | BYTE | bitmap as defined in [4] |
| 3 |  | bTimeOut2 | BYTE | Does IFD distinguish bTimeOut from bTimeOut2? |
| 0 | IFD does not distinguish bTimeOut from bTimeOut2 |
| 1 | IFD distinguishes bTimeOut from bTimeOut2 |
| 4  (optional) |  | bAdvancedFlags | BYTE | Collection of advanced PIN flags  (not present implies no advanced support) |
| 0 | bAdaptiveFrameSize |  | Does IFD support Adaptive PIN Frame size: |
| 0 | No: IFD does not support Adaptive PIN Frame size |
| 1 | Yes: IFD supports Adaptive PIN Frame size |
| 1 | bAdvancedModify |  | Does IFD support Advanced Modify structure: |
| 0 | No: IFD does not support Advanced Modify structure |
| 1 | Yes: IFD supports Advanced Modify structure |
| 2..7 |  |  | RFU |

### DISPLAY\_PROPERTIES

|  |  |  |  |
| --- | --- | --- | --- |
| **Byte offset** | **Field** | **Type** | **Description** |
| 0 | wLcdMaxCharacters | USHORT | Maximum number of characters on a single line |
| 2 | wLcdMaxLines | USHORT | Maximum number of lines that can be used |

### SET\_SPE\_MESSAGE

|  |  |  |  |
| --- | --- | --- | --- |
| **Byte offset** | **Field** | **Type** | **Description** |
| 0 | bApplicationId | BYTE[32] | Unique application ID |
| 32 | bMessageIndex | BYTE | Index of the message which should be set |
| 33 | wLangId | USHORT | Language ID for message |
| 35 | bMessageLength | BYTE | Length of message, in bytes |
| 36 | bMessage | BYTE[] | Message string in UTF-8 |

### PIN\_VERIFY\_APP\_ID

|  |  |  |  |
| --- | --- | --- | --- |
| **Byte offset** | **Field** | **Type** | **Description** |
| 0 | bApplicationId | BYTE [32] | unique application ID |
| 32 | bTimeOut | BYTE | timeout in seconds (00 means use default timeout) |
| 33 | bTimeOut2 | BYTE | timeout in seconds after first key stroke |
| 34 | bmFormatString | BYTE | formatting options USB\_CCID\_PIN\_FORMAT\_xxx |
| 35 | bmPINBlockString | BYTE | bits 7-4 bit size of PIN length in APDU  bits 3-0 PIN block size in bytes after justification and formatting |
| 36 | bmPINLengthFormat | BYTE | bits 7-5 RFU, bit 4 set if system units are bytes clear if system units are bits,  bits 3-0 PIN length position in system units |
| 37 | wPINMaxExtraDigit | USHORT | XXYY, where XX is minimum PIN size in digits, YY is maximum |
| 39 | bEntryValidationCondition | BYTE | Conditions under which PIN entry should be  considered complete |
| 40 | bNumberMessage | BYTE | Number of messages to display for PIN verification |
| 41 | wLangId | USHORT | Language for messages |
| 43 | bMsgIndex | BYTE | Message index (should be 00) |
| 44 | bTeoPrologue | BYTE[3] | T=1 I-block prologue field to use (fill with 00) |
| 47 | ulDataLength | ULONG | length of Data to be sent to the ICC |
| 51 | abData | BYTE[] | Data to send to the ICC |

Detailed information about each structure element (except of bApplicationId and bimeout2) can be found in reference [4].

### PIN\_MODIFY\_APP\_ID

| **Byte offset** | **Field** | **Type** | **Description** |
| --- | --- | --- | --- |
| 0 | bApplicationId | BYTE[32] | unique application ID |
| 32 | bTimeOut | BYTE | timeout in seconds (00 means use default timeout) |
| 33 | bTimeOut2 | BYTE | timeout in seconds after first key stroke |
| 34 | bmFormatString | BYTE | formatting options USB\_CCID\_PIN\_FORMAT\_xxx |
| 35 | bmPINBlockString | BYTE | bits 7-4 bit size of PIN length in APDU  bits 3-0 PIN block size in bytes after justification and formatting |
| 36 | bmPINLengthFormat | BYTE | bits 7-5 RFU, bit 4 set if system units are bytes clear if system units are bits,  bits 3-0 PIN length position in system units |
| 37 | bInsertionOffsetOld | BYTE | Insertion position offset in bytes for the current PIN |
| 38 | bInsertionOffsetNew | BYTE | Insertion position offset in bytes for the new PIN |
| 39 | wPINMaxExtraDigit | USHORT | XXYY, where XX is minimum PIN size in digits, YY is maximum |
| 41 | bConfirmPIN | BYTE | Flags governing need for confirmation of new PIN |
| 42 | bEntryValidationCondition | BYTE | Conditions under which PIN entry should be  considered complete |
| 43 | bNumberMessage | BYTE | Number of messages to display for PIN verification |
| 44 | wLangId | USHORT | Language for messages |
| 46 | bMsgIndex1 | BYTE | Index of 1st prompting message |
| 47 | bMsgIndex2 | BYTE | Index of 2nd prompting message |
| 48 | bMsgIndex3 | BYTE | Index of 3rd prompting message |
| 49 | bTeoPrologue | BYTE[3] | T=1 I-block prologue field to use (fill with 00) |
| 52 | ulDataLength | ULONG | length of Data to be sent to the ICC |
| 56 | abData | BYTE[] | Data to send to the ICC |

Detailed information about each structure element (except of bApplicationId and bTimeout2) can be found in [4].

### WRITE\_DISPLAY

|  |  |  |  |
| --- | --- | --- | --- |
| **Byte offset** | **Field** | **Type** | **Description** |
| 0 | wDisplayTime | USHORT | Display time in ms |
| 2 | bPosX | BYTE | Column (starting at 0) |
| 3 | bPosY | BYTE | Row (starting at 0) |
| 4 | wLangId | USHORT | Language ID of the message |
| 6 | bStringLength | BYTE | Length of message, in bytes |
| 7 | bString | BYTE[] | message string in UTF-8 |

bPosX and bPosY must within the boundaries specified by FEATURE\_IFD\_DISPLAY\_PROPERTIES.

### GET\_KEY

|  |  |  |  |
| --- | --- | --- | --- |
| **Byte offset** | **Field** | **Type** | **Description** |
| 0 | wWaitTime | USHORT | Time in seconds to wait for a key to be hit |
| 2 | bMode | BYTE | Display mode of key |
| 3 | bPosX | BYTE | Column (starting at 0) |
| 4 | bPosY | BYTE | Row (starting at 0) |

Following values are possible for bMode:

|  |  |
| --- | --- |
| 0 | Character of key is displayed |
| 1 | Asterisk (\*) is displayed for each hit key |
| 2 | Nothing is displayed |
| All others | RFU |

### EXECUTE\_PACE

The InBuffer structure is defined in the following table:

| Byte offset | Field | Type | Description |
| --- | --- | --- | --- |
| 0 | idxFunction | BYTE | Index of PACE function 1 – GetReaderPACECapabilities 2 – EstablishPACEChannel 3 – DestroyPACEChannel |
| 1 | length\_InputData | USHORT | Length of InputData |
| 3 | InputData | BYTE[] | Depends on function (Field idxFunction) |

The OutBuffer structure is defined in the in the next table:

|  |  |  |  |
| --- | --- | --- | --- |
| Byte offset | Field | Type | Description |
| 0 | Result | ULONG | See table below (result values) |
| 4 | length\_OutputData | USHORT | Length of OutputData |
| 3 | OutputData | BYTE[] | Depends on function (see InBuffer) |

Table for result values:

|  |  |
| --- | --- |
| ***Result*** | ***Description*** |
| 0x00000000 | No error |
| Errors in input data | |
| 0xD0000001 | Inconsistent lengths in input |
| 0xD0000002 | Unexpected data in input |
| 0xD0000003 | Unexpected combination of data in input |
| Errors during protocol execution | |
| 0xE0000001 | Syntax error in TLV response |
| 0xE0000002 | Unexpected or missing object in TLV response |
| 0xE0000003 | Unknown PIN-ID |
| 0xE0000006 | Wrong Authentication Token |
| Response APDU of the card reports error (status code SW1SW2) | |
| 0xF000SW1SW2 | Select EF.CardAccess |
| 0xF001SW1SW2 | Read Binary EF.CardAccess |
| 0xF002SW1SW2 | MSE: Set AT |
| 0xF003SW1SW2 | General Authenticate Step 1 |
| 0xF004SW1SW2 | General Authenticate Step 2 |
| 0xF005SW1SW2 | General Authenticate Step 3 |
| 0xF006SW1SW2 | General Authenticate Step 4 |
| Others | |
| 0xF0100001 | Communication abort (e.g. card removed during protocol) |
| 0xF0100002 | No card |
| 0xF0200001 | Abort |
| 0xF0200002 | Timeout |
| Additional application dependent error codes may be used. | |

## Feature list

### FEATURE\_VERIFY\_PIN\_START / FEATURE\_MODIFY\_PIN\_START

FEATURE\_VERIFY\_PIN\_START uses the PIN\_VERIFY for the input buffer.

FEATURE\_MODIFY\_PIN\_START uses the PIN\_MODIFY for the input buffer.

The IFD handler will return no data back to Control function.

Implementation Note for Windows:

In case of a parameter error of the passed structure, the SCardControl may return ERROR\_INVALID\_PARAMETER (0x57).

### FEATURE\_GET\_KEY\_PRESSED

After the control code for FEATURE\_VERIFY\_PIN\_START or FEATURE\_MODIFY\_PIN\_START has been sent to the IFD handler, the control code for FEATURE\_GET\_KEY\_PRESSED can be used to determine if a key has been pressed.

The IFD handler will send one byte back to the Control function according to following table.

|  |  |
| --- | --- |
| ‘0’– ‘9’ button (valid keys) | 0x2B |
| Cancel button | 0x1B |
| Correction/Backspace button | 0x08 |
| Enter/Ok button | 0x0d |
| Timeout finished SPE operation | 0x0e |
| No key since last call | 0x00 |
| PIN\_Operation\_Aborted | 0x40 (used e.g. for timeout indication, parameter error) |
| all keys cleared (by backspace) | 0x0a (This value can be returned optionally) |

Applications can use the returned information to display feedback to the user (e.g. pseudo display). This control code must be used in a polling mode.

Implementation notes for

1. Validation condition is equal to ‘Timeout occurred’:

The IFD handler may send 0x0d or optionally 0x0e . 0x0e can be used to give the calling application the possibility to distinguish which user operation finished the SPE operation if the validation condition is a combination of ‘Timeout occurred’ + ‘Validation key pressed’.

1. Validation condition is equal to ‘ Max size reached’

The IFD handler must send for each valid key a 0x2B back to the calling control function. The Ok/Enter button must be ignored for the whole SPE operation. If the max PIN size is reached, the IFD handler must not send a 0x0d to indicate that the SPE operation has finished. The application itself knows when the maximum PIN size has been reached and therefore needs to indication.

### FEATURE\_VERIFY\_PIN\_FINISH / FEATURE\_MODIFY\_PIN\_FINISH

The control code for the FEATURE\_VERIFY\_PIN\_FINISH or FEATURE\_MODIFY\_PIN\_FINISH must be used to retrieve the final result from the secure pin entry operation.

The IFD handler will return 2 bytes according to following table:

|  |  |
| --- | --- |
| 64 00 | SPE operation timed out |
| 64 01 | SPE operation was cancelled by the ‘Cancel’ button |
| 64 02 (1) | Modify PIN operation failed because two “new PIN” entries do not match |
| 64 03 | User entered too short or too long PIN regarding MIN/MAX PIN Length  Note: as this error code is not known by CT-API implementations, it should be mapped to 64 01 on CT-API level. |
| 6b 80 | invalid parameter in passed structure |
| SW1 SW2 | result from the card |

1) 64 02 occurs if the user enters two different “new PIN’s” during the Modify PIN operation. In that case, no Change PIN command has been sent to the smart card

### FEATURE\_VERIFY\_PIN\_DIRECT / FEATURE\_MODIFY\_PIN\_DIRECT

The control codes for these features use the same PIN structures which are already described in sections 2.5.2 and 2.5.3.

The main purposes for these control codes are:

1. class 2 readers without display for which no user feedback to the host is either required or supported.
2. class 2 readers with display which is used for SPE messages and user feedback.

The IFD handler will return the same responses as described for the FEATURE\_VERIFY\_PIN\_FINISH/ FEATURE\_MODIFY\_PIN\_FINISH.

### FEATURE\_ABORT

This control code for FEATURE\_ABORT can be used to cancel any of the actions initiated for the following control codes:

- FEATURE\_VERIFY\_PIN\_START

- FEATURE\_MODIFY\_PIN\_START

The control codes for FEATURE\_VERIFY\_PIN\_DIRECT and FEATURE\_MODIFY\_PIN\_DIRECT will NOT require this abort.

This control code may be required for devices, which do not have a display, as end users may use a CANCEL button at the host application to cancel the SPE.

This control code will always succeed. Also, after this ABORT control code, there is no need for applications to use the control code for FEATURE\_VERIFY\_PIN\_FINISH and FEATURE\_MODIFY\_PIN\_FINISH, respectively, as the same 2 bytes as returned by the finish operation can be returned by this ABORT control code.

The IFD handler will return 2 bytes according to the following table:

|  |  |
| --- | --- |
| 64 80 | SPE operation was aborted by the a ‘Cancel’ operation at the host system |
| SW1 SW2 | result from the card |

### FEATURE\_MCT\_READER\_DIRECT

The control code for FEATURE\_MCT\_READER\_DIRECT can be used to transmit a command to the IFD.

The IFD handler will return a buffer containing data (data size can be null) and two status bytes SW1 SW2 according to table 12 of [1] and chapter 4.2 of [3].

### FEATURE\_MCT\_UNIVERSAL

This control code for FEATURE\_MCT\_UNIVERSAL can be used to transmit a command to the IFD or to any of the ICCs of the IFD.

Therefore, a MCT\_UNIVERSAL structure is passed to the IFD handler. The SAD and DAD fields have to be filled with values according to table 6 of document [2]. The Buffer field contains the command APDU.

The IFD handler will return a MCT\_UNIVERSAL structure with SAD and DAD fields containing values concerning to table 7 of [2].

The buffer field will contain response data and two status bytes SW1 SW2 according to table 12 of [1] and chapter 4.2 of [3].

### FEATURE\_IFD\_PIN\_PROPERTIES

This feature can be used – if supported by the IFD handler – to retrieve the properties of the IFD regarding PIN handling:

1. The IFD handler returns the size of a possible display as described in [4]
2. The IFD handler returns which entry validation conditions are supported as described in [4]
3. The IFD handler returns if the reader distinguishes between bTimeOut from bTimeOut2

The input parameter for this feature is a NULL pointer.

The output parameter is a pointer to a PIN\_PROPERTIES structure.

Note: wLcdLayout is also used to indicate if a display is present. If wLcdLayout = 0x0000, the IFD has no display.

If an IFD handler does not support this feature, an application must assume following default values:

wLcdLayout = 0x0000 no display present

bEntryValidationCondition = 0x07 timeout reached, max PIN size reached, validation key pressed

bTimeOut2 = 0x00 IFD does not distinguish bTimeOut from TimeOut2

### FEATURE\_IFD\_DISPLAY\_PROPERTIES

The IFD handler returns the number of characters that can be displayed on a single line. If this is greater than the physical number of characters, the IFD is capable of scrolling messages that exceed the physical characteristics. The scrolling behavior is specific to the IFD.

The IFD handler returns the number of lines that can be used to display custom message strings. This number may be less than the physical number of lines (ex. if the IFD wants to reserve some space for hard coded messages – say, “Enter PIN”).

The input parameter for this feature is a NULL pointer.

The output parameter is a pointer to a DISPLAY\_PROPERTIES structure.

If an IFD handler does not support this feature, an application must assume following default values:

wLcdMaxCharacters = 0x0000 and wLcdMaxLines = 0x0000: no display is present

### FEATURE\_SET\_SPE\_MESSAGE

This feature can be used to define a message which should be displayed during any SPE operation by an IFD with display. After storing the SPE message in the IFD, the application just needs to properly set either bMsgIndex (FEATURE\_VERIFY\_PIN\_DIRECT) or bMsgIndex1, bMsgIndex2 and bMsgIndex3 (FEATURE\_MODIFY\_PIN\_DIRECT) to get the message displayed during any SPE operation.

The message is stored in the IFD until the next power on reset (e.g. reboot, plug off- plug in). Optionally IFDs may also store application dependent SPE messages permanently.

The application must provide following information in the SET\_SPE\_MESSAGE structure:

* bApplicationId:

For each application different SPE strings may be used. Otherwise applications will get conflicts if the same message index is used but the SPE message has a different meaning. bApplicationId should be unique and can completely defined by any PC/SC application itself.

* wLanguageId:

This must correspond to an ANSI code page that should be used for any conversion. If this is set to zero, then the device may choose any code page which may result in loss of data.

* bMessageIndex:

Index of the SPE message

* bMessageLength

length of the SPE message, in bytes

* bMessage

buffer which contains SPE message in UTF-8

The IFD must provide for each application ID and for each language ID a storage for up to 254 messages. In case the IFD cannot store a message, the IFD handler must return response code Out\_Of\_Memory.

If the message index is not within the range of 0x00 – 0xFE, the IFD handler must return the response code Device\_Wrong\_Parameter. The value 0xFF is reserved for further purposes.

Applications can use FEATURE\_IFD\_PIN\_PROPERTIES to retrieve the display capabilities of the IFD.

If an application sets a message which is longer than wLcdMaxCharacters, following 2 options are allowed:

* 1. The IFD returns Device\_Wrong\_Parameter
  2. The IFD displays as many characters as possible and cuts off the message string. In this case Device\_Success must be returned.

An application can use the carriage return character (0x0d) to control which part of the message should be displayed in the first and in the second line.

It is not possible to set more than one message by a single call to the IFD. Each new message requires a separate call.

The IFD MAY convert bMessage into a native format using the wLanguageId before storing.

The default SPE messages, which do not depend on any application and which are IFD vendor specific, cannot be overwritten. Whenever an application uses 2.6.4, these message are displayed.

### FEATURE\_VERIFY\_PIN\_DIRECT\_APP\_ID / FEATURE\_MODIFY\_PIN\_DIRECT\_APP\_ID

These features can be used as the features described in 2.6.4, but additionally a unique application ID will be passed to the IFD which is necessary to display the application specific SPE messages which have been set by 2.6.10.

### FEATURE\_WRITE\_DISPLAY

This feature can be used to write any UTF-8 based message to the display if SPE is not active.

The input buffer is a pointer to a WRITE\_DISPLAY structure.

The output buffer is a NULL pointer.

The parameter display time has following meaning :

|  |  |
| --- | --- |
| wDisplayTime = 0 | Message is displayed forever or until a new message is written again |
| wDisplayTime > 0 | Message is displayed as long as specified. After the time has elapsed, a terminal specific message is display again (idle message) |

Warning:

There is a potential security hazard when supporting this feature; any string may be easily displayed by any PC/SC application, including a malware.

Combination of 2.6.12 and 2.6.13 features may lead to a malware requesting a false PIN request and retrieving the PIN code.

### FEATURE\_GET\_KEY

This feature can be used to retrieve the value of a pressed key if SPE is not active.

The GET\_KEY structure is used for the input buffer, the output buffer holds just one byte for the pressed key.

The application get following values for a pressed keys:

|  |  |
| --- | --- |
| Key | Returned Value |
| 0 | 0x30 |
| 1 | 0x31 |
| 2 | 0x32 |
| 3 | 0x33 |
| 4 | 0x34 |
| 5 | 0x35 |
| 6 | 0x36 |
| 7 | 0x37 |
| 8 | 0x38 |
| 9 | 0x39 |
| \* | 0x2A |
| . | 0x2E |
| Cancel | 0x1b |
| Backspace | 0x08 |
| Menu | 0x4D (‘M’) |
| OK | 0x0d |

Warning:

There is a potential security hazard when supporting this feature; the keys entered by using this GET\_KEY feature at the Secure PIN IFD may be easily grabbed by any PC/SC application, including a malware.

Combination of 2.6.12 and 2.6.13 features may lead to a malware requesting a false PIN request and retrieving the PIN code.

### FEATURE\_GET\_TLV\_PROPERTIES

This feature can be used to retrieve IFD properties in TLV form.

In order to add flexibility and avoid potential inconsistencies in data structures returned by the IFD (e.g. when a new data field in needed in a structure such as PIN\_PROPERTIES), a new approach using TLV fields (Tag Length Value) is available through this feature.

This serves several objectives:

* 1. Retrieve all field properties structures (such as PIN\_PROPERTIES and DISPLAY\_PROPERTIES) via an unique feature request
  2. Allow new fields to be added without breaking the existing feature specification

The input parameter for this feature is a NULL pointer.

The output parameter is a pointer to a TLV oriented structure as follows:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Tag#1 | Length#1 | Value#1 | … | Tag#n | Length#n | Value#n |

Tag and Length are coded on 1 byte.

Value depends on the Tag itself (see table below).

Values representing an integer greater than 1 byte are little-endian byte-ordered.

| **Tag** | **Length** | **Value** |
| --- | --- | --- |
| 0x00 | - | Reserved for future use. |
| 0x01 | 2 | wLcdLayout. See [4]. |
| 0x02 | 1 | bEntryValidationCondition. See [4]. |
| 0x03 | 1 | bTimeOut2. See [4]. |
| 0x04 | 2 | wLcdMaxCharacters. Maximum number of characters on a single line. See 2.5.6. |
| 0x05 | 2 | wLcdMaxLines. . Maximum number of lines that can be used. See 2.5.6 |
| 0x06 | 1 | bMinPINSize. Minimum PIN size accepted by the reader. |
| 0x07 | 1 | bMaxPINSize. Maximal PIN size accepted by the reader. |
| 0x08 | n | sFirmwareID. String using UTF-8 format indicating the reader firmware string. |
| 0x09 | 1 | bPPDUSupport.  Bit0: If set to 1, PPDU is supported over SCardControl using FEATURE\_CCID\_ESC\_COMMAND  Bit1: If set to 1, PPDU is supported over SCardTransmit |
| 0x0A | 4 | dwMaxAPDUDataSize  Maximal size of data the reader and its driver can support  0: short APDU only.  0<X<=256: forbidden values (RFU)  256 < X <= 0x10000: short and extended APDU of up to X bytes of data  0x10000 < X: invalid values (RFU) |
| 0x0B | 2 | wIdVendor  USB Vendor ID |
| 0x0C | 2 | wIdProduct  USB Product ID |
| 0x0D | 2 \* n | An array of n elements, each element (USHORT) is a wLangId. This wLangId represents a selectable language for the functions PIN\_VERIFY and PIN\_MODIFY.  The first element represents the ‘default language’. |
| 0x0D…0xFF | - | Reserved for future use. |

### FEATURE\_CCID\_ESC\_COMMAND

This feature can be used to retrieve the control code to send a CCID escape command (PC\_to\_RDR\_Escape see [4]) to the reader.

The input parameter for this feature is a pointer to the abData field (see [4]) containing the specific escape command.

The output parameter is a pointer to a buffer that will contain the reader response.

Note that a CCID escape command is specific to a given reader, so before issuing this command, the application has to make sure it addresses the appropriate reader.

### FEATURE\_EXECUTE\_PACE

This new feature is defined for support of PACE. The PACE protocol is a secure channel between the IFD and the smart card. PACE is described in [TR-03110] and [ICAO].

This chapter describes only how an application can trigger the IFD PACE protocol.

***GetReaderPACECapabilities***

The command GetReaderPACECapabilities is used to query the PACE support of the IFD. The result is given as a bit mask.

* 0x80 denotes support of explicit DestroyPACEChannel 1);
* 0x40 denotes generic PACE support;

Other bits denote the support of application specific extensions of the protocol, e.g.

* 0x20 denotes support for the eID application of the German eID-Card,
* 0x10 support for the qualified electronic signature function on contactless cards.

These extensions may comprise support for specialized coding of further parameters, using a display of the IFD or performing additional cryptographic functions.

1) Note: Not all IFD’s support this feature, because these IFD’s destroy the PACE channel implicitly after the transaction.

InputData: None.

|  |  |  |  |
| --- | --- | --- | --- |
| ***Number*** | ***Type*** | ***Name*** | ***Description*** |
| 1 | BYTE | length\_BitMap | Length of BitMap |
| 2 | BYTE[] | BitMap | 0x40 – The IFD supports PACE 0x80 – The IFD supports DestroyPACEChannel other – Application specific extensions supported |

OutputData:

***EstablishPACEChannel***

The command EstablishPACEChannel is used to trigger the execution of the PACE protocol between the chip and the IFD. The result of this command is the verification of the PIN and the establishment of a secure messaging channel between the chip and the IFD. Until secure messaging is stopped, the IFD must encrypt/decrypt APDUs received from the host/the chip before forwarding them to the chip/the host, respectively.

Note: If no PIN is given in InputData, the PIN must be entered on the secure PIN entry device of the IFD by the user. The key Kπ is derived from the PIN for use in PACE.

Input data of EstablishPACEChannel is defined as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| ***Number*** | ***Type*** | ***Name*** | ***Description*** |
| 1 | BYTE | PinID | 0x01: MRZ 0x02: CAN 0x03: PIN 0x04: PUK |
| The following elements are only present if the execution of PACE is to be followed by an execution of Terminal Authentication Version 2 as defined in **Error! Reference source not found.**.**Error! Reference source not found.** | | | |
| 2 | BYTE | length\_CHAT | Length of CHAT |
| 3 | BYTE[] | CHAT | Role Identifier and Certificate Holder Authorization Template, see **Error! Reference source not found.**. |
| If the PIN to be used is not secret (e.g. printed on the card/stored in the host), it may be delivered by the host to the IFD in the following elements. A suitable command filter should be employed by the IFD to refuse delivery of secret PINs by the host. | | | |
| 4 + Len | BYTE | length\_PIN | Length of PIN |
| 5 + Len | BYTE[] | PIN | Password given by the host |
| Further application dependent data may follow. Depending on the data additional actions may be performed by the IFD during the execution of EstablishPACEChannel. | | | |

Output data of EstablishPACEChannel is defined as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| ***Number*** | ***Type*** | ***Name*** | ***Description*** |
| 1 | USHORT | Status | Status codes of MSE:Set AT |
| 2 | USHORT | length\_CardAccess | Length of EF\_CardAccess |
| 3 | BYTE[] | EF\_CardAccess | Contents of EF.CardAccess as read from the chip |
| The following elements are only present if the execution of PACE is to be followed by an execution of Terminal Authentication Version 2 as defined in [TR-03110]. These data are needed to perform the Terminal Authentication. | | | |
| 4 | BYTE | length\_CARcurr | Length of CARcurr |
| 5 | BYTE[] | CARcurr | Current Certificate Authority Reference |
| 6 | BYTE | length\_CARprev | Length of CARprev |
| 7 | BYTE[] | CARprev | Previous Certificate Authority Reference |
| 8 | USHORT | length\_IDicc | Length of IDicc |
| 9 | BYTE[] | IDicc | Ephemeral PACE public key of the IFD |

***DestroyPACEChannel***

The command DestroyPACEChannel terminate the PACE channel.

Note: Not all IFD’s support this feature, because these IFD’s destroy the PACE channel implicitly after the transaction.

InputData: None.

OutputData: None.

# Feature Readers

## System Architecture

This section deals with feature readers and their integration into the PC/SC architecture.

Feature IFD

***ICC***

***ICC***

ICC Resource Manager

ICC Aware Applications

IFD

Handler

IFD

Handler

IFD

Handler

ICC Service Provider

IFD Service Provider

SPE for class 2/3

readers

Feature IFD

***ICC***

Feature IFD

Figure 2- General Architecture

## Definition of Features

### General Description

Chipcard readers are becoming more intelligent: features as secure PIN entry are becoming very important. This part of the PC/SC specifications defines general features of the subsystem.

A feature is defined by its *Feature Number* and the accompanying *Feature Command Data* and *Feature Response Data*.

An application can query the subsystem which features are supported. In the response, the application receives a list of *Feature Numbers*; this list represents all supported features on the subsystem.

A feature is executed on the IFD by commanding it through a Pseudo-APDU (PPDU)

### Feature Execution

A certain feature is represented by its *Feature Number*; the input data for a feature is presented in the accompanying *Feature Command Data*.

The result of the feature’s execution is presented in the *Feature Response Data*.

Feature

Response

Data

Feature

Feature

Command

Data

#### Feature Execution by Pseudo-APDU

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| command header | | | | command body | |
| CLA | INS | P1 | P2 | Lc | Command |
| ‘FF’ | ‘C2’ | ‘01’ | *Feature*  *Number* | Lc | *Feature Command Data* |

This Pseudo-APDU is defined as a command header (CLA/INS/P1/P2) and an optional command body, according to [*5*], chapter 12.

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

|  |  |  |
| --- | --- | --- |
| response data | response status  SW1/SW2 | description |
| *Feature Response Data* | 90 00 | Feature executed successful,  *Feature Response Data*  is present |
| -empty- | 6A 86 | Incorrect value for P2  (requested feature not present) |

This response is defined as an optional ‘response data’ part plus a 2-byte status code, in line with [*5*], chapter 12.

This Pseudo-APDU is represented by a number of sequential bytes (a buffer), this shall be exchanged by means of the Transmit method (see [*6*]), as follows:

**RESPONSECODE Transmit(**

IN SCARD\_IO\_HEADER SendPci // Send protocol structure

IN BYTE[] SendBuffer // Data buffer for send data

IN OUT SCARD\_IO\_HEADER RecvPci // Receive protocol structure

IN OUT BYTE[] RecvBuffer // Data buffer for receive data

OUT DWORD RecvLength // Length of received data

)

The SendPci must contain the protocol structure of the current inserted card.

The SendBuffer contains the *Feature Command Data*.

The RecvPci contains the protocol structure used to communicate.

The RecvBuffer will contain the *Feature Response Data* when the subsystem has successfully executed this command. A successful execution is marked when the response status SW1/SW2 has value ’90 00’.

A note on endianness:

For a successful use of the Pseudo-APDU interface, values in the structures of [ref 7] representing an integer greater than 1 byte are defined to be ordered low-byte first (little-endian byte-order). Remind that in [ref 7] the byte ordering is decided by machine architecture.

### Get List of Features (GET\_FEATURE\_REQUEST)

A reader (subsystem) may contain a certain number of features. The application shall be able to request the actual supported feature(s) of the current subsystem.

The following features are currently defined:

|  |  |
| --- | --- |
| **Feature** | **Feature Number** |
| FEATURE\_VERIFY\_PIN\_START | 0x01 |
| FEATURE\_VERIFY\_PIN\_FINISH | 0x02 |
| FEATURE\_MODIFY\_PIN\_START | 0x03 |
| FEATURE\_MODIFY\_PIN\_FINISH | 0x04 |
| FEATURE\_GET\_KEY\_PRESSED | 0x05 |
| FEATURE\_VERIFY\_PIN\_DIRECT | 0x06 |
| FEATURE\_MODIFY\_PIN\_DIRECT | 0x07 |
| FEATURE\_MCT\_READER\_DIRECT | 0x08 |
| FEATURE\_MCT\_UNIVERSAL | 0x09 |
| FEATURE\_IFD\_PIN\_PROPERTIES | 0x0A |
| FEATURE\_ABORT | 0x0B |
| FEATURE\_SET\_SPE\_MESSAGE | 0x0C |
| FEATURE\_VERIFY\_PIN\_DIRECT\_APP\_ID | 0x0D |
| FEATURE\_MODIFY\_PIN\_DIRECT\_APP\_ID | 0x0E |
| FEATURE\_WRITE\_DISPLAY | 0x0F |
| FEATURE\_GET\_KEY | 0x10 |
| FEATURE\_IFD\_DISPLAY\_PROPERTIES | 0x11 |
| FEATURE\_GET\_TLV\_PROPERTIES | 0x12 |
| FEATURE\_CCID\_ESC\_COMMAND | 0x13 |
| FEATURE\_EXECUTE\_PACE | 0x20 |

Table 1. Feature Numbers

#### GET\_FEATURE\_REQUEST by Pseudo-APDU

The GET\_FEATURE\_REQUEST returns all features in the IFD (see also chapter 2).

This PPDU feature shall execute (see ch 3.2.2.1) as follows:

* The *FeatureNumber* is 0x00.
* The *FeatureCommandData* is empty (*Lc* = 0).
* The *FeatureResponseData* is a byte array: each byte in this array represents a feature number present in this reader. Table 1 defines the FeatureNumbers.

## Features

### FEATURE\_VERIFY\_PIN\_START

The FEATURE\_VERIFY\_PIN\_START starts an indirect PIN procedure in the IFD (see also chapter 2).

This PPDU feature shall execute (see chapter 3.2.2) as follows:

* The *FeatureNumber* is 0x01.
* The *FeatureCommandData* conforms to the PIN\_VERIFY structure, chapter 2.5.2.
* the *FeatureResponseData* is empty.

See also the other indirect PIN features FEATURE\_GET\_KEY\_PRESSED, FEATURE\_VERIFY\_PIN\_FINISH and FEATURE\_ABORT.

### FEATURE\_VERIFY\_PIN\_FINISH

The FEATURE\_VERIFY\_PIN\_FINISH ends an indirect PIN procedure in the IFD (see also chapter 2).

This PPDU feature shall execute (see chapter 3.2.2) as follows:

* The *FeatureNumber* is 0x02.
* The *FeatureCommandData* is empty.
* the *FeatureResponseData* is a 2-byte status conforming to chapter 2.6.3.

See also the other indirect PIN features FEATURE\_VERIFY\_PIN\_START, FEATURE\_ABORT, FEATURE\_GET\_KEY\_PRESSED.

### FEATURE\_MODIFY\_PIN\_START

The FEATURE\_MODIFY\_PIN\_FINISH starts an indirect PIN procedure in the IFD (see also chapter 2).

This PPDU feature shall execute (see chapter 3.2.2) as follows:

* The *FeatureNumber* is 0x03.
* The *FeatureCommandData* conforms to the PIN\_MODIFY structure, chapter 2.5.3.
* the *FeatureResponseData* is empty.

See also the other indirect PIN features FEATURE\_MODIFY\_PIN\_FINISH, FEATURE\_ABORT, FEATURE\_GET\_KEY\_PRESSED.

### FEATURE\_MODIFY\_PIN\_FINISH

The FEATURE\_MODIFY\_PIN\_FINISH ends an indirect PIN procedure in the IFD (see also chapter 2).

This PPDU feature shall execute (see chapter 3.2.2) as follows:

* The *FeatureNumber* is 0x04.
* The *FeatureCommandData* is empty.
* the *FeatureResponseData* is a 2-byte status conforming to chapter 2.6.3.

See also the other indirect PIN features FEATURE\_MODIFY\_PIN\_START, FEATURE\_ABORT, FEATURE\_GET\_KEY\_PRESSED.

### FEATURE\_GET\_KEY\_PRESSED

The FEATURE\_GET\_KEY\_PRESSED can be used at an indirect PIN procedure in the IFD (see also chapter 2).

This PPDU feature shall execute (see chapter 3.2.2) as follows:

* The *FeatureNumber* is 0x05.
* The *FeatureCommandData* is empty.
* the *FeatureResponseData* is a single byte conforming to chapter 2.6.2.

See also the other indirect PIN features FEATURE\_VERIFY\_PIN\_START, FEATURE\_VERIFY\_PIN\_FINISH, FEATURE\_MODIFY\_PIN\_START, FEATURE\_MODIFY\_PIN\_FINISH, FEATURE\_ABORT.

### FEATURE\_VERIFY\_PIN\_DIRECT

The FEATURE\_VERIFY\_PIN\_DIRECT performs a complete (direct) PIN procedure in the IFD (see also chapter 2).

This PPDU feature shall execute (see chapter 3.2.2) as follows:

* The *FeatureNumber* is 0x06.
* The *FeatureCommandData* conforms to the PIN\_VERIFY structure, chapter 2.5.2.
* the *FeatureResponseData* is a 2-byte status conforming to chapter 2.6.4.

### FEATURE\_MODIFY\_PIN\_DIRECT

The FEATURE\_MODIFY\_PIN\_DIRECT performs a complete (direct) PIN procedure in the IFD (see also chapter 2).

This PPDU feature shall execute (see chapter 3.2.2) as follows:

* The *FeatureNumber* is 0x07.
* The *FeatureCommandData* conforms to the PIN\_MODIFY structure, chapter 2.5.3.
* the *FeatureResponseData* is a 2-byte status conforming to chapter 2.6.4.

### FEATURE\_MCT\_READER\_DIRECT

The FEATURE\_MCT\_READER\_DIRECT can be used to transmit a command to the IFD (see also chapter 2).

This PPDU feature shall execute (see chapter 3.2.2) as follows:

* The *FeatureNumber* is 0x08.
* The *FeatureCommandData* is vendor specific, see chapter 2.6.6.
* the *FeatureResponseData* is a buffer containing vendor specific data (data size can be null), see chapter 2.6.6.

### FEATURE\_MCT\_UNIVERSAL

The FEATURE\_MCT\_UNIVERSAL can be used to transmit a command to the IFD or the ICC (see also chapter 2).

This PPDU feature shall execute (see chapter 3.2.2) as follows:

* The *FeatureNumber* is 0x09.
* The *FeatureCommandData* conforms to the MCT\_UNIVERSAL structure, see chapter 2.5.4.
* the *FeatureResponseData* will contain data conforming to the MCT\_UNIVERSAL structure with SAD and DAD fields containing values concerning to table 7 of [2].

### FEATURE\_IFD\_PIN\_PROPERTIES

The FEATURE\_IFD\_PIN\_PROPERTIES can be used to retrieve the properties of the IFD regarding PIN handling (see also chapter 2).

This PPDU feature shall execute (see chapter 3.2.2) as follows:

* The *FeatureNumber* is 0x0A.
* The *FeatureCommandData* is empty.
* the *FeatureResponseData* conforms to the PIN\_PROPERTIES structure (see chapter 2.5.5).

### FEATURE\_ABORT

The FEATURE\_ABORT aborts an indirect PIN procedure in the IFD (see also chapter 2).

This PPDU feature shall execute (see chapter 3.2.2) as follows:

* The *FeatureNumber* is 0x0B.
* The *FeatureCommandData* is empty.
* the *FeatureResponseData* is a 2-byte status conforming to chapter 2.6.5.

See also the other indirect PIN features FEATURE\_GET\_KEY\_PRESSED, FEATURE\_VERIFY\_PIN\_START, FEATURE\_VERIFY\_PIN\_FINISH, FEATURE\_MODIFY\_PIN\_START, FEATURE\_MODIFY\_PIN\_FINISH.

### FEATURE\_SET\_SPE\_MESSAGE

The FEATURE\_SET\_SPE\_MESSAGE can be used to define a message which should be displayed during an SPE operation in the IFD (see also chapter 2).

This PPDU feature shall execute (see chapter 3.2.2) as follows:

* The *FeatureNumber* is 0x0C.
* The *FeatureCommandData* conforms to the SET\_SPE\_MESSAGE structure, chapter 2.5.7.
* the *FeatureResponseData* contains a 2-byte status conforming to chapter 2.6.10.

See also the SPE related features FEATURE\_VERIFY\_PIN\_DIRECT, FEATURE\_VERIFY\_PIN\_DIRECT\_APP\_ID, FEATURE\_MODIFY\_PIN\_DIRECT, FEATURE\_MODIFY\_PIN\_DIRECT\_APP\_ID.

### FEATURE\_VERIFY\_PIN\_DIRECT\_APP\_ID

The FEATURE\_VERIFY\_PIN\_DIRECT\_APP\_ID performs a complete (direct) PIN procedure in the IFD (see also chapter 2), based on specific SPE messages.

This PPDU feature shall execute (see chapter 3.2.2) as follows:

* The *FeatureNumber* is 0x0D.
* The *FeatureCommandData* conforms to the PIN\_VERIFYAPP\_ID structure, chapter 2.5.8.
* the *FeatureResponseData* is a 2-byte status conforming to chapter 2.6.3.

See also the SPE related features FEATURE\_SET\_SPE\_MESSAGE, FEATURE\_MODIFY\_PIN\_DIRECT\_APP\_ID.

### FEATURE\_MODIFY\_PIN\_DIRECT\_APP\_ID

The FEATURE\_MODIFY\_PIN\_DIRECT\_APP\_ID performs a complete (direct) PIN procedure in the IFD (see also chapter 2), based on specific SPE messages.

This PPDU feature shall execute (see chapter 3.2.2) as follows:

* The *FeatureNumber* is 0x0E.
* The *FeatureCommandData* conforms to the PIN\_MODIFY\_APP\_ID structure, chapter 2.5.9.
* the *FeatureResponseData* is a 2-byte status conforming to chapter 2.6.3.

See also the SPE related features FEATURE\_SET\_SPE\_MESSAGE, FEATURE\_VERIFY\_PIN\_DIRECT\_APP\_ID.

### FEATURE\_WRITE\_DISPLAY

The FEATURE\_WRITE\_DISPLAY writes any UTF-8 based message on the display of the IFD (see also chapter 2), if SPE is not active.

This PPDU feature shall execute (see chapter 3.2.2) as follows:

* The *FeatureNumber* is 0x0F.
* The *FeatureCommandData* conforms to the WRITE\_DISPLAY structure, chapter 2.5.10.
* the *FeatureResponseData* is empty.

### FEATURE\_GET\_KEY

The FEATURE\_GET\_KEY retrieves the value of a pressed key on the keypad of the IFD (see also chapter 2), if SPE is not active.

This PPDU feature shall execute (see chapter 3.2.2) as follows:

* The *FeatureNumber* is 0x10.
* The *FeatureCommandData* conforms to the GET\_KEY structure, chapter 2.5.11.
* the *FeatureResponseData* is a single byte conforming to chapter 2.6.13.

### FEATURE\_IFD\_DISPLAY\_PROPERTIES

The FEATURE\_IFD\_DISPLAY\_PROPERTIES returns a structure with the properties of the display of the IFD (see also chapter 2).

This PPDU feature shall execute (see chapter 3.2.2) as follows:

* The *FeatureNumber* is 0x11.
* The *FeatureCommandData* is empty.
* the *FeatureResponseData* conforms to the DISPLAY\_PROPERTIES structure, chapter 2.5.6.

### FEATURE\_GET\_TLV\_PROPERTIES

The FEATURE\_GET\_TLV\_PROPERTIES returns a TLV list of the properties of the IFD (see also chapter 2).

This PPDU feature shall execute (see chapter 3.2.2) as follows:

* The *FeatureNumber* is 0x12.
* The *FeatureCommandData* is empty.
* the *FeatureResponseData* is a TLV structure conforming to chapter 2.6.14.

### FEATURE\_CCID\_ESC\_COMMAND

The FEATURE\_CCID\_ESC\_COMMAND is used to exchange vendor proprietary information with the reader (see also chapter 2).

This PPDU feature shall execute (see chapter 3.2.2) as follows:

* The *FeatureNumber* is 0x13.
* The *FeatureCommandData* is vendor specific.
* the *FeatureResponseData* is vendor specific.

### FEATURE\_EXECUTE\_PACE

The FEATURE\_EXECUTE\_PACE is used to command the PACE functionality within the reader (see also 2.6.16).

This PPDU feature shall execute (see chapter 3.2.2) as follows:

* The *FeatureNumber* is 0x20.
* The *FeatureCommandData* conforms to the InBuffer structure of chapter 2.6.16.
* the *FeatureResponseData* conforms to the OutBuffer structure of chapter 2.6.16.

# Abbreviations

APDU Application Protocol Data Unit

CAN Card Access Number

CAR Certification Authority Reference

EF Elementary File

IFD Interface Device

MCT Multifunctional Card Terminal

MRZ Machine Readable Zone

PACE Password Authenticated Connection Establishment

PIN Personal Identification Number

PPDU Peripheral Processor Data Unit (Pseudo-APDU)

PUK PIN Unblock Key

SPE Secure PIN Entry

TLV Tag Length Value

References

[1] International technology – Identification cards – Integrated circuit(s) cards with contacts – Part 4: Inter-industry commands for interchange; International Standard ISO/IEC 7816-4:1995(E)

[2] Multifunctional Card Terminals, Part 3 - Application Independent Card Terminal Application Programming Interface for ICC Applications (CT-API 1.1); Deutsche Telekom AG / PZ Telesec, SIT Fraunhofer Institut für Sichere Telekooperation, TÜV Informationstechnik GmbH, TeleTrust Deutschland e.V.; 2002

[3] Multifunctional Card Terminals, Part 4 - CT-BCS - Application Independent Card Terminal Basic Command Set; Deutsche Telekom AG / PZ Telesec, SIT Fraunhofer Institut für Sichere Telekooperation, TÜV Informationstechnik GmbH, TeleTrust Deutschland e.V.; 2002

[4] USB Serial Bus Device Class Spec of USB Chip/Smart Card Interface Devices, Revision 1.1

[5] ISO/IEC 7816: Identification cards — Integrated circuit cards — Part 3: Cards with contacts — Electrical interface and transmission protocols, ISO/IEC 7816-3 Third edition 2006-11-01

[6] Interoperability Specification for ICCs and Personal Computer Systems, Part 5. ICC Resource Manager Definition, Revision 2.01.01 September 2005

[TR-03110] Technical Guideline TR-03110, Advanced Security Mechanisms for Machine Readable Travel Documents - Extended Access Control (EAC), Password Authenticated Connection Establishment (PACE) and Restricted Identification (RI), <https://www.bsi.bund.de/ElektronischeAusweiseTR>

[ICAO] ICAO: Technical Report "Supplemental Access Control", <http://mrtd.icao.int>