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

Revision 2.02.08

April 2010

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	January 2012	Xiring: Added Extended APDU Support property as a new TLV property (dwMaxAPDUDataSize). Added Reader Model property as two new TLV properties (wIdVendor and wIdProduct).

Contents

1 System Architecture

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

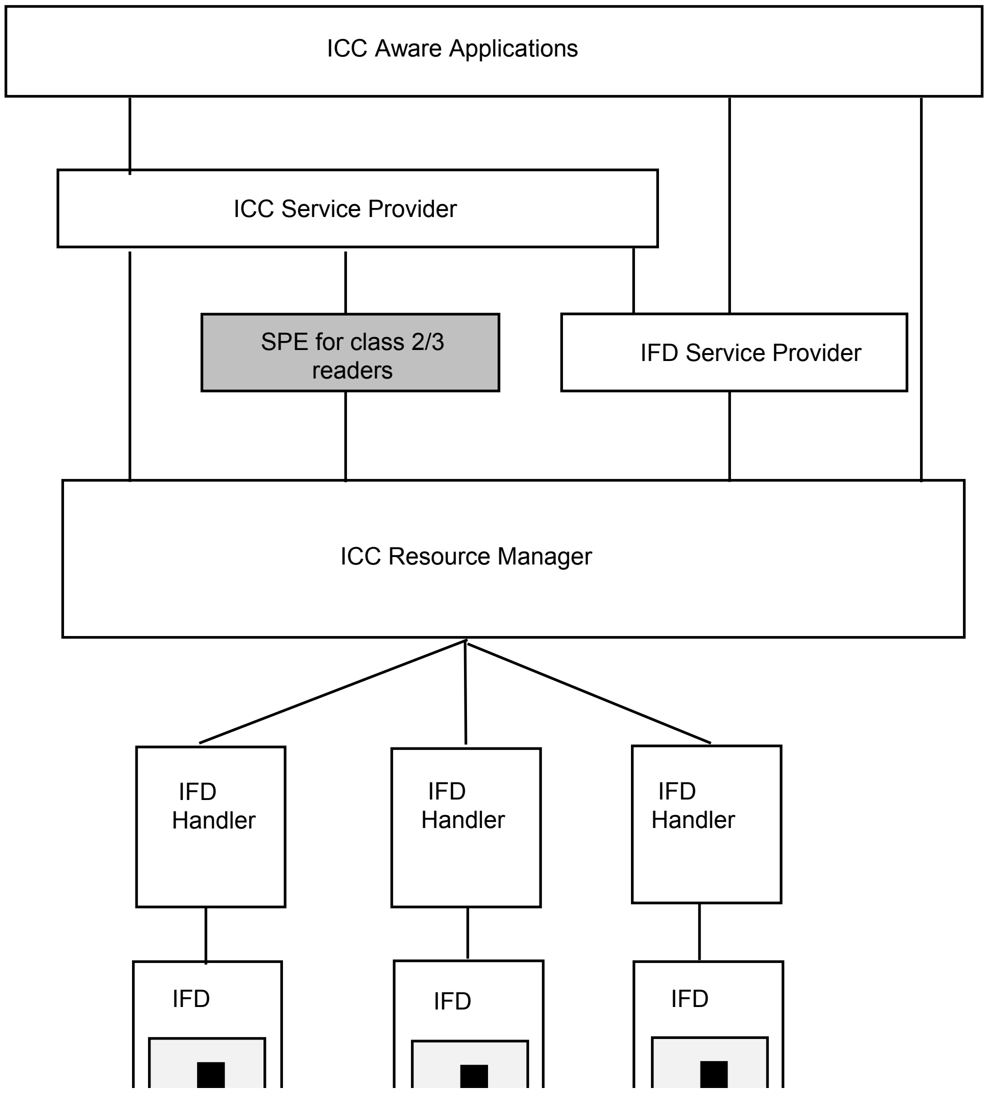

Figure - General Architecture

Definition of Features ~~and Control Codes~~

1.1 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 queries 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.

There are two methods to execute a feature:
(1) method Control Code
(2) method Pseudo-APDU (PPDU)

1.2 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 Structure*.

1.2.1 Feature Execution by Control Code

A Feature Number shall be converted into a Control Code, by using the TLV list as specified in ch GET_FEATURE_REQUEST by Control Code.
A control code is 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.

```
RESPONSECODE Control (
        IN              DWORD           ControlCode,
        IN              BYTE[]          InBuffer,
        IN OUT          BYTE[]          Out Buffer,
        OUT             DWORD           OutBufferLength
        )
```

ControlCode represents the feature to be executed.
InBuffer represents the byte values of the *Feature Command Data*
OutBuffer represents the byte values of the *Feature Response Data*

Implementation Note for Windows:
Under Windows 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.

1.2.2 Feature Execution by Pseudo-APDU

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 [*ISO7816-3*], chapter 12.

Any Pseudo-APDU command will always generate a response:

response	response status
	SW1/SW2
Feature Response Data	

This response is defined as an optional 'response data' part plus a 2-byte status code, in line with [*ISO7816-3*], 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 [*PCSCp5*]), 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* after the subsystem has executed this command.

1.3 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	**Tag value**
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

Table

1.3.1 GET_FEATURE_REQUEST by Control Code

This feature shall execute (see ch Feature Execution by Control Code) as follows:

- The *ControlCode* is ~~The corresponding control code is~~ 3400 decimal.
- The *InBuffer* is empty.
-
- ~~It is mandatory for class 2 drivers to support this control code.~~

~~**Out:**~~

- ~~The driver shall return Device_Success and~~the *OutBuffer* is 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 ~~c~~Control ~~c~~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.

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

01 04 XX XX XX XX 02 04 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

e.g. an IFD handler which supports none of the features will return 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)
```

1.3.2 GET_FEATURE_REQUEST by Pseudo-APDU

This feature shall execute (see ch Feature Execution by Pseudo-APDU) as follows:

- The *FeatureNumber* is 0.
- The *InBuffer* is empty.
- the *OutBuffer* is a byte array: each byte in this array represents a feature number present in this reader. Table defines the FeatureNumbers.

~~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.~~

1.4 ~~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)~~

1.5 ~~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~~

1.6 Type definitions

In thi~~s~~se following chapters, these data types are ~~defined~~used ~~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.

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

~~01 04 XX XX XX XX 02 04 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~~

~~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.~~

1.7 ~~Function Call~~

~~RESPONSECODE Control (~~
~~IN DWORD ControlCode,~~
~~IN BYTE[] InBuffer,~~
~~IN OUT BYTE[] Out Buffer,~~
~~OUT DWORD OutBufferLength~~
~~)~~

~~Implementation Note for Windows:~~
~~Under Windows 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.~~

2 F~~Structure list~~eature structures

2.1 PIN_VERIFY structure

Byte offset	Field	Type	Description
0	bTimeOut	BYTE	timeout in seconds (00 means use default timeout)
1	bTimeOut2	BYTE	timeout in seconds after first key stroke
2	bmFormatString	BYTE	formatting options USB_CCID_PIN_FORMAT_xxx
3	bmPINBlockString	BYTE	bits 7-4 bit size of PIN length in APDU bits 3-0 PIN block size in bytes after justification and formatting
4	bmPINLengthFormat	BYTE	bits 7-5 RFU, bit 4 set if system units are bytes clear if system units are bits, bits 3-0 PIN length position in system units

5	wPINMaxExtraDigit	USHORT	XXYY, where XX is minimum PIN size in digits, YY is maximum
7	bEntryValidationCondition	BYTE	Conditions under which PIN entry should be considered complete
8	bNumberMessage	BYTE	Number of messages to display for PIN verification
9	wLangId	USHORT	Language for messages
11	bMsgIndex	BYTE	Message index (should be 00)
12	bTeoPrologue	BYTE[3]	T=1 I-block prologue field to use (fill with 00)
15	ulDataLength	ULONG	length of Data to be sent to the ICC
19	abData	BYTE[]	Data to send to the ICC

Table

Detailed information about each structure element can be found in [4].

2.2 PIN_MODIFY structure

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

Table

Detailed information about each structure element can be found in [4].

2.3 MCT_UNIVERSAL structure

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

Table

2.4 PIN_PROPERTIES structure

Byte offset	Field	Type	Description
0	wLcdLayout	USHORT	display characteristics as defined in [4]
2	bEntryValidationCondition	BYTE	bitmap as defined in [4]
3	bTimeOut2	BYTE	0 = IFD does not distinguish bTimeOut from bTimeOut2 1 = IFD distinguishes bTimeOut from bTimeOut2

Table

2.5 DISPLAY_PROPERTIES structure

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

Table

2.6 SET_SPE_MESSAGE structure

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

Table

2.7 PIN_VERIFY_APP_ID structure

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

Table

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

2.8 PIN_MODIFY_APP_ID structure

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

Table

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

2.9 WRITE_DISPLAY structure

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

Table

bPosX and bPosY must within the boundaries specified by FEATURE_IFD_DISPLAY_PROPERTIES.

2.10 GET_KEY structure

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)

Table

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

Table

3 Features

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

This feature shall execute (see ch Feature Execution) as follows:

- The *FeatureNumber* is 0x01.
- The *InBuffer* is according Table .
- the *OutBuffer* is a two-byte array representing the status SW1/SW2.

See also FEATURE_GET_KEY_PRESSED, FEATURE_VERIFY_PIN_FINISH and FEATURE_ABORT

3.2 FEATURE_VERIFY_PIN_FINISH

This feature shall execute (see ch Feature Execution) as follows:

- The *FeatureNumber* is 0x02.
- The *InBuffer* is empty.
- the *OutBuffer* is according 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

Table

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

See also FEATURE_VERIFY_PIN_START, FEATURE_ABORT, FEATURE_GET_KEY_PRESSED

3.3 FEATURE_MODIFY_PIN_START

This feature shall execute (see ch Feature Execution) as follows:

- The *FeatureNumber* is 0x03.
- The *InBuffer* is accordingTable .
- the *OutBuffer* is a two-byte array representing the status SW1/SW2.

See also FEATURE_MODIFY_PIN_FINISH, FEATURE_ABORT, FEATURE_GET_KEY_PRESSED

3.4 FEATURE_MODIFY_PIN_FINISH

This feature shall execute (see ch Feature Execution) as follows:

- The *FeatureNumber* is 0x04.
- The *InBuffer* is empty.
- the *OutBuffer* is according Table .

See also FEATURE_MODIFY_PIN_START, FEATURE, FEATURE_ABORT, FEATURE_GET_KEY_PRESSED

3.5 FEATURE_GET_KEY_PRESSED

This feature shall execute (see ch Feature Execution) as follows:

- The *FeatureNumber* is 0x05.
- The *InBuffer* is empty.
- the *OutBuffer* is according Table , followed by a status SW1/SW2

‘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)

Table

See also FEATURE_VERIFY_PIN_START, FEATURE_VERIFY_PIN_FINISH, FEATURE_MODIFY_PIN_START, FEATURE_MODIFY_PIN_FINISH, FEATURE_ABORT

3.6 FEATURE_VERIFY_PIN_DIRECT

This feature shall execute (see ch Feature Execution) as follows:
- The *FeatureNumber* is 0x06.
- The *InBuffer* is according Table .
- the *OutBuffer* is according Table .

The main purposes for this feature is:
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.

3.7 FEATURE_MODIFY_PIN_DIRECT

This feature shall execute (see ch Feature Execution) as follows:
- The *FeatureNumber* is 0x07.
- The *InBuffer* is according Table .
- the *OutBuffer* is according Table .

The main purposes for this feature is:
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.

3.8 FEATURE_MCT_READER_DIRECT

This feature shall execute (see ch Feature Execution) as follows:
- The *FeatureNumber* is 0x08.
- The *InBuffer* is vendor specific.
- the *OutBuffer* is a buffer containing vendor specific data (data size can be null) and two status bytes SW1/SW2 according to table 12 of [1] and chapter 4.2 of [3].

The FEATURE_MCT_READER_DIRECT can be used to transmit a command to the subsystem.

3.9 FEATURE_MCT_UNIVERSAL

This feature shall execute (see ch Feature Execution) as follows:

- The *FeatureNumber* is 0x09.
- The *InBuffer* is according Table .
- the *OutBuffer* will contain a MCT_UNIVERSAL structure (see Table) with SAD and DAD fields containing values concerning to table 7 of [2], followed by two status bytes SW1 SW2 according to table 12 of [1] and chapter 4.2 of [3]..

The FEATURE_MCT_UNIVERSAL can be used to transmit a command to the subsystem or to any of the ICCs of the subsystem.

Therefore a MCT_UNIVERSAL structure is passed. 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.

3.10 FEATURE_IFD_PIN_PROPERTIES

This feature shall execute (see ch Feature Execution) as follows:

- The *FeatureNumber* is 0x0A.
- The *InBuffer* is empty.
- the *OutBuffer* is according Table , followed by two status bytes SW1/SW2 according to table 12 of [1] and chapter 4.2 of [3].

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

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

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

If a subsystem 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	subsystem does not distinguish bTimeOut from TimeOut2

3.11 FEATURE_ABORT

This feature shall execute (see ch Feature Execution) as follows:

- The *FeatureNumber* is 0x0B.
- The *InBuffer* is empty.
- the *OutBuffer* is according Table

64 80	SPE operation was aborted by the a 'Cancel' operation at the host system
SW1 SW2	result from the card

Table

The FEATURE_ABORT can be used to cancel any of the actions initiated for the following features:
- FEATURE_VERIFY_PIN_START
- FEATURE_MODIFY_PIN_START

The FEATURE_VERIFY_PIN_DIRECT and FEATURE_MODIFY_PIN_DIRECT will NOT require this abort.
This FEATURE_ABORT 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 feature will always succeed. Also, after this ABORT executed, there is no need for applications to use the FEATURE_VERIFY_PIN_FINISH or FEATURE_MODIFY_PIN_FINISH, respectively, as the same 2 bytes as returned by the finish operation can be returned by this FEATURE_ABORT.

See also FEATURE_GET_KEY_PRESSED, FEATURE_VERIFY_PIN_START, FEATURE_VERIFY_PIN_FINISH, FEATURE_MODIFY_PIN_START, FEATURE_MODIFY_PIN_FINISH

3.12 FEATURE_SET_SPE_MESSAGE

This feature shall execute (see ch Feature Execution) as follows:

- The *FeatureNumber* is 0x0C.
- The *InBuffer* is according Table .
- the *OutBuffer* contains two status bytes SW1/SW2 according to table 12 of [1] and chapter 4.2 of [3].

This feature can be used to define a message which should be displayed during any SPE operation by a subsystem with display. After storing the SPE message in the subsystem, 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 subsystem until the next power on reset (e.g. reboot, plug off- plug in). Optionally subsystems 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 subsystem must provide for each application ID and for each language ID a storage for up to 254 messages. In case the subsystem cannot store a message, the subsystem must return response code Out_Of_Memory.

If the message index is not within the range of 0x00 – 0xFE, the subsystem 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 :

a. The IFD returns Device_Wrong_Parameter

b. 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 subsystem. Each new message requires a separate call.

The subsystem 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 vendor specific, cannot be overwritten. Whenever an application uses FEATURE_VERIFY_PIN_DIRECT or FEATURE_MODIFY_PIN_DIRECT, these message are displayed.

3.13 FEATURE_VERIFY_PIN_DIRECT_APP_ID

This feature shall execute (see ch Feature Execution) as follows:
- The *FeatureNumber* is 0x0D.
- The *InBuffer* is according Table .
- the *OutBuffer* is accordingTable .

This feature can be used as the FEATURE_VERIFY_PIN_DIRECT, but additionally an unique application ID will be passed to the subsystem which is necessary to display the application specific SPE messages which have been set by FEATURE_SET_SPE_MESSAGE.

3.14 FEATURE_MODIFY_PIN_DIRECT_APP_ID

This feature shall execute (see ch Feature Execution) as follows:

- The *FeatureNumber* is 0x0E.
- The *InBuffer* is according Table .
- the *OutBuffer* is according Table

This feature can be used as the FEATURE_MODIFY_PIN_DIRECT, but additionally an unique application ID will be passed to the subsystem which is necessary to display the application specific SPE messages which have been set by FEATURE_SET_SPE_MESSAGE.

3.15 FEATURE_WRITE_DISPLAY

This feature shall execute (see ch Feature Execution) as follows:

- The *FeatureNumber* is 0x0F.
- The *InBuffer* is according Table .
- the *OutBuffer* is two status bytes SW1/SW2 according to table 12 of [1] and chapter 4.2 of [3].

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

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 FEATURE_WRITE_DISPLAY and FEATURE_GET_KEY may lead to a malware requesting a false PIN request and retrieving the PIN code.

3.16 FEATURE_GET_KEY

This feature shall execute (see ch Feature Execution) as follows:

- The *FeatureNumber* is 0x10.
- The *InBuffer* is according Table .
- the *OutBuffer* is a single byte according Table , followed by two status bytes SW1/SW2 according to table 12 of [1] and chapter 4.2 of [3].

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

Table

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.

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 FEATURE_WRITE_DISPLAY and FEATURE_GET_KEY may lead to a malware requesting a false PIN request and retrieving the PIN code.

3.17 FEATURE_IFD_DISPLAY_PROPERTIES

This feature shall execute (see ch Feature Execution) as follows:

- The *FeatureNumber* is 0x11.
- The *InBuffer* is empty.
- the *OutBuffer* is according Table

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

The subsystem 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 subsystem wants to reserve some space for hard coded messages – say, “Enter PIN”).

If a subsystem does not support this feature, an application must assume following default values:
wLcdMaxCharacters = 0x0000 and wLcdMaxLines = 0x0000 : no display is present

3.18 FEATURE_GET_TLV_PROPERTIES

This feature shall execute (see ch Feature Execution) as follows:

- The *FeatureNumber* is 0x12.
- The *InBuffer* is empty.
- the *OutBuffer* is a TLV structure based on the elements in Table , followed by two status bytes SW1 SW2 according to table 12 of [1] and chapter 4.2 of [3].

Tag (1 byte)	**Length** (1 byte)	**Value** (little endian)
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 DISPLAY_PROPERTIES.
0x05	2	wLcdMaxLines. . Maximum number of lines that can be used. See DISPLAY_PROPERTIES
0x06	1	bMinPINSize. Minimum PIN size accepted by the reader.
0x07	1	bMaxPINSize. Maximal PIN size accepted by the reader.

Tag (1 byte)	**Length** (1 byte)	**Value** (little endian)
0x08	n	sFirmwareID. String using UTF-8 format indicating the reader firmware string.
0x09	1	(RFU)
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… 0xFF	-	Reserved for future use.

Table

This feature can be used to retrieve subsystem properties in TLV form.
In order to add flexibility and avoid potential inconsistencies in data structures returned by the subsystem (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:

a. Retrieve all field properties structures (such as PIN_PROPERTIES and DISPLAY_PROPERTIES) via an unique feature request
b. Allow new fields to be added without breaking the existing feature specification

3.19 FEATURE_CCID_ESC_COMMAND

This feature shall execute (see ch Feature Execution) as follows:

- The *FeatureNumber* is 0x13.
- The *InBuffer* is vendor specific.
- the *OutBuffer* is vendor specific

This feature can be used to perform a CCID escape command (PC_to_RDR_Escape see [4]) to the reader.

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.

3.19.1 ~~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.~~

3.19.2 ~~PIN_VERIFY~~

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

~~Detailed information about each structure element can be found in [4].~~

3.19.3 ~~PIN_MODIFY~~

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

~~Detailed information about each structure element can be found in [4].~~

3.19.4 ~~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~~

3.19.5 ~~PIN_PROPERTIES~~

~~Byte offset~~	~~Field~~	~~Type~~	~~Description~~
~~0~~	~~wLcdLayout~~	~~USHORT~~	~~display characteristics as defined in [4]~~
~~2~~	~~bEntryValidationCondition~~	~~BYTE~~	~~bitmap as defined in [4]~~
~~3~~	~~bTimeOut2~~	~~BYTE~~	~~0 = IFD does not distinguish bTimeOut from bTimeOut2~~ ~~1 = IFD distinguishes bTimeOut from bTimeOut2~~

3.19.6 ~~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~~

3.19.7 ~~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~~

3.19.8 ~~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 [4].~~

3.19.9 ~~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].~~

3.19.10 ~~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.~~

3.19.11 ~~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~~

3.20 ~~Feature list~~

3.20.1 ~~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).~~

3.20.2 ~~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 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~~

2) ~~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'.~~

3) ~~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.~~

3.20.3 ~~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~~

3.20.4 ~~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~~ PIN_V~~ and~~ PIN_~~.~~

~~The main purposes for these control codes are:~~

3) ~~class 2 readers without display for which no user feedback to the host is either required or supported.~~

4) ~~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.~~

3.20.5 ~~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 following table:~~

~~64 80~~	~~SPE operation was aborted by the a 'Cancel' operation at the host system~~
~~SW1 SW2~~	~~result from the card~~

3.20.6 ~~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].~~

3.20.7 ~~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].~~

3.20.8 ~~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 :~~

4) ~~The IFD handler returns the size of a possible display as described in [4]~~

5) ~~The IFD handler returns which entry validation conditions are supported as described in [4]~~

6) ~~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~~

3.20.9 ~~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~~

3.20.10 ~~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 :~~

a. ~~The IFD returns Device_Wrong_Parameter~~

b. ~~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~~ FEATURE_VERIFY_PIN_DIRECT / FEATURE_MODIFY_PIN_DIRECT~~, these message are displayed.~~

3.20.11 ~~FEATURE_VERIFY_PIN_DIRECT_APP_ID / FEATURE_MODIFY_PIN_DIRECT_APP_ID~~

~~These features can be used as the features described in~~ FEATURE_VERIFY_PIN_DIRECT / FEATURE_MODIFY_PIN_DIRECT~~, but additionally an unique application ID will be passed to the IFD which is necessary to display the application specific SPE messages which have been set by~~ FEATURE_SET_SPE_MESSAGE~~.~~

3.20.12 ~~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~~ FEATURE_WRITE_DISPLAY ~~and~~ FEATURE_GET_KEY ~~features may lead to a malware requesting a false PIN request and retrieving the PIN code.~~

3.20.13 ~~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~~ FEATURE_WRITE_DISPLAY ~~and~~ FEATURE_GET_KEY ~~features may lead to a malware requesting a false PIN request and retrieving the PIN code.~~

1.1.1 ~~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:~~

a. ~~Retrieve all field properties structures (such as PIN_PROPERTIES and DISPLAY_PROPERTIES) via an unique feature request~~
b. ~~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~~ DISPLAY_PROPERTIES~~.~~
~~0x05~~	~~2~~	~~wLcdMaxLines. . Maximum number of lines that can be used. See~~ DISPLAY_PROPERTIES
~~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~~

~~Tag~~	~~Length~~	~~Value~~
~~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~~
~~0x0DA…~~ ~~0xFF~~	~~-~~	~~Reserved for future use.~~

1.1.2 ~~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.~~
~~(values from~~ Erreur : source de la référence non trouvée~~)~~

Abbreviations

IFD	Interface Device
MCT	Multifunctional Card Terminal
PIN	Personal Identification Number
SPE	Secure PIN Entry
TLV	Tag Length Value
PPDU	Peripheral Processor Data Unit

References

[1] International technology – Identification cards – Integrated circuit(s) cards with contacts – Part 4: Interindustry 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

[*ISO7816-3*]

[*PCSCp5*]