INTERNATIONAL STANDARD

ISO/IEC 7816-12

First edition
2005-10-01

Identification cards — Integrated circuit cards —

Part 12: Cards with contacts — USB electrical interface and operating procedures

Cartes d'identification — Cartes à circuit intégré —

Partie 12: Cartes à contacts — Interface électrique USB et procédures de fonctionnement

Reference number
ISO/IEC 7816-12:2005(E)

PDF disclaimer

This PDF file may contain embedded typefaces. In accordance with Adobe's licensing policy, this file may be printed or viewed but shall not be edited unless the typefaces which are embedded are licensed to and installed on the computer performing the editing. In downloading this file, parties accept therein the responsibility of not infringing Adobe's licensing policy. The ISO Central Secretariat accepts no liability in this area.

Adobe is a trademark of Adobe Systems Incorporated.

Details of the software products used to create this PDF file can be found in the General Info relative to the file; the PDF-creation parameters were optimized for printing. Every care has been taken to ensure that the file is suitable for use by ISO member bodies. In the unlikely event that a problem relating to it is found, please inform the Central Secretariat at the address given below.

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Contents

Page

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

ISO/IEC 7816-12 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 17, *Cards and personal identification*.

ISO/IEC 7816 consists of the following parts, under the general title *Identification cards — Integrated circuit cards*:

— *Part 1: Cards with contacts — Physical characteristics*

— *Part 2: Cards with contacts — Dimensions and location of the contacts*

— *Part 3: Cards with contacts — Electrical interface and transmission protocols*

— *Part 4: Organization, security and commands for interchange*

— *Part 5: Registration of application providers*

— *Part 6: Interindustry data elements for interchange*

— *Part 7: Interindustry commands for Structured Card Query Language (SCQL)*

— *Part 8: Commands for security operations*

— *Part 9: Commands for card management*

— *Part 10: Cards with contacts — Electronic signals and answer to reset for synchronous cards*

— *Part 11: Personal verification through biometric methods*

— *Part 12: Cards with contacts — USB electrical interface and operating procedures*

— *Part 15: Cryptographic information application*

ISO/IEC 10536[2] specifies access by close coupling. ISO/IEC 14443[3] and 15693[4] specify access by radio frequency. Such cards are also known as contactless cards.

Introduction

ISO/IEC 7816 is a series of documents specifying integrated circuit cards and the use of such cards for interchange. These cards are identification cards intended for information exchange negotiated between the outside world and the integrated circuit in the card. As a result of an information exchange, the card delivers information (computation result, stored data), and / or modifies its content (data storage, event memorization).

— Five parts are specific to cards with galvanic contacts and three of them specify electrical interfaces.

- ISO/IEC 7816-1 specifies physical characteristics for cards with contacts.
- ISO/IEC 7816-2 specifies dimensions and location of the contacts.
- ISO/IEC 7816-3 specifies electrical interface and transmission protocols for asynchronous cards.
- ISO/IEC 7816-10 specifies electrical interface and answer to reset for synchronous cards.
- ISO/IEC 7816-12 specifies electrical interface and operating procedures for USB cards.

— All the other parts are independent from the physical interface technology. They apply to cards accessed by contacts and / or by radio frequency.

- ISO/IEC 7816-4 specifies organization, security and commands for interchange.
- ISO/IEC 7816-5 specifies registration of application providers.
- ISO/IEC 7816-6 specifies interindustry data elements for interchange.
- ISO/IEC 7816-7 specifies commands for structured card query language.
- ISO/IEC 7816-8 specifies commands for security operations.
- ISO/IEC 7816-9 specifies commands for card management.
- ISO/IEC 7816-11 specifies personal verification through biometric methods.
- ISO/IEC 7816-15 specifies cryptographic information application.

The International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC) draw attention to the fact that it is claimed that compliance with this document may involve the use of the following patents:

WO 00/16255, *Data transmission method and card therefor*, 23 March 2000

Declared for ISO/IEC 7816-2

WO 01/69881, *A method of communication between a smart card and a host station*, 20 September 2001

WO 01/57684 A1, *Conveying protocol units for portable electronic objects via a protocol for microcomputer peripherals*, 9 August 2001

0001399 / France, *Transport d'unités de protocole d'objet électronique portable par protocole pour péripheriques de micro-ordinateur*

09/775668 / USA, *Conveying protocol units for portable electronic objects via a protocol for microcomputer peripherals*

1904043 / Europe, *Transport d'unités de protocole d'objet électronique portable par protocole pour péripheriques de micro-ordinateur*

1804474 / China, *Conveying protocol units for portable electronic objects via a protocol for microcomputer peripherals*

PCT / FR01 / 00326, *Transport d'unités de protocole d'objet électronique portable par protocole pour péripheriques de micro-ordinateur*

US 6148354*, Architecture for a universal serial bus-based PC flash disk*

US 6763399, *USB key apparatus for interacting with a USB host via a USB port*

ISO and IEC take no position concerning the evidence, validity and scope of these patent rights.

The holders of these patent rights have assured the ISO and IEC that they are willing to negotiate licences under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statements of the holders of these patent rights are registered with the ISO and IEC. Information may be obtained from:

Contact	Patent number
Schlumberger Systèmes, France	WO 00/16255 WO 01/69881
GEMPLUS, France	WO 01/57684 A1 0001399 / France / Granted 09/775668 / USA / Pending 1904043 / Europe / Pending 1804474 / China / Pending PCT / FR01 / 00326 / Pending
M-Systems, Israel	US 6148354
Aladdin Knowledge Systems, USA	US 6763399

Infineon Technologies has not identified any patents but confirms that it is prepared to license its patents, both granted and pending, which may be deemed necessary to manufacture, use, and sell implementations of ISO/IEC 7816-12 on reasonable and non-discretionary terms and conditions.

The following companies may hold patents relating to this part of ISO/IEC 7816 but have not provided details of the patents or agreed to provide licenses:

Orga Kartensysteme GmbH, Germany	AU 752627
Renesas, Japan	US 20050052924 US 20040070952
ST Microelectronics	US 6769622 WO 02/317161

Attention is drawn to the possibility that some elements of the document may be the subject of patent rights other than those identified above. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

Identification cards — Integrated circuit cards —

Part 12:
Cards with contacts — USB electrical interface and operating procedures

1 Scope

This part of ISO/IEC 7816 specifies the operating conditions of an integrated circuit card that provides a USB interface. Figure 1 shows the assignment of the contact fields for a USB interface and – to illustrate interoperability – the assignment as used in ISO/IEC 7816-3.

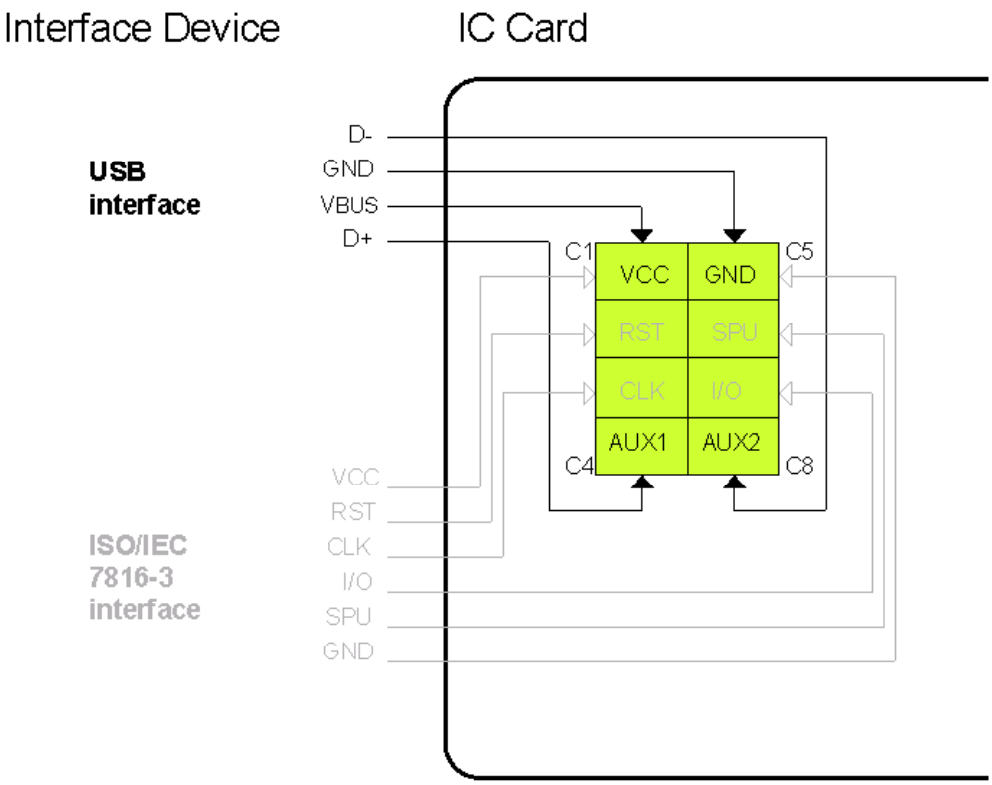


Figure 1 — Assignment of contacts for a USB integrated circuit card

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 7816-2:1999/Amd.1:2004, *Identification cards — Integrated circuit cards — Part 2: Cards with contacts — Dimensions and location of the contacts — Amendment 1: Assignment of contacts C4 and C8*

ISO/IEC 7816-3, *Identification cards — Integrated circuit cards — Part 3: Cards with contacts — Electrical interface and transmission protocols*

Universal Serial Bus Specification Revision 2.0, April 27, 2000
USB Implementers Forum
Available at <http://www.usb.org/developers/docs>

Universal Serial Bus, Device Class Specification for
USB Chip/Smart Card Interface Devices , Revision 1.00, March 20, 2001
USB Implementers Forum, Device Working Group: Smart Card
Available at <http://www.usb.org/developers/devclass_docs>

3 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

3.1 Device

3.1.1
interface device
terminal communication device or machine to which the card is electrically connected during operation [ISO/IEC 7816-3]

3.1.2
USB connection device
device providing an electrical connection path between a USB-ICC and a USB host or hub

3.2 Terms and definitions used in other specifications

For the purposes of this document, the terms and definitions given in the USB specification and the CCID specification (see Clause 4) apply.

NOTE The relevant terms used in this document are listed in informative Annexes C and D.

4 Abbreviations and notation

For the purposes of this document, the following abbreviations apply.

Protocol T=0, Protocol T=1 [ISO/IEC 7816-3]
D+, D- [Universal Serial Bus Specification Revision 2.0]

USB specification
Referencing to Universal Serial Bus Specification Revision 2.0 (see clause 2).

CCID
Chip Card Interface Device. Designates an interface device controlled via USB.

CCID specification
Reference to the Device Class Specification for USB Chip/Smart Card Interface Devices (see clause 2).

USB-ICC
USB Integrated Circuit Card. An integrated circuit card providing a USB interface.

5 Electrical characteristics of the contacts

The assignment of the contacts for USB operating conditions is given in ISO/IEC 7816-2:1999 and ISO/IEC 7816-2:1999/Amd.1:2004.

An interface device will provide a USB connection to a USB-ICC through VCC, GND, AUX1 and AUX2 respectively VBUS, GND, D+ and D- defined by the USB specification.

Cards designed for ISO/IEC 7816-3 operating conditions shall not be damaged when activated under USB conditions. Conversely, cards designed for USB operation shall not be damaged when activated under ISO/IEC 7816-3 operating conditions (by definition, a damaged card no longer operates as specified or contains corrupt data).

6 USB-ICC operated by an interface device

A USB-ICC that only provides a USB interface shall have electrically connected C1, C5, C4 and C8. All other contact fields shall be electrically isolated. This type of USB-ICC can be operated by a USB connection device. The USB connection device shall establish an electrical connection to C1, C5, C4 and C8 only, following the electrical characteristics and protocol given in the USB specification.

An interface device that does not support a USB interface shall have AUX1 and AUX2 electrically isolated or ensure that the voltage applied at these contact fields shall remain between –0,3V and Vcc + 0,3V.

7 USB Descriptors

7.1 Standard Descriptors

The standard descriptors described in the USB specification constitute a way for the host software to identify a new USB device attached, and to load one or more appropriate drivers for this new USB device. The standard descriptors are read by the host software during the enumeration process. In addition, the descriptors can also be retrieved by the host software using standard USB requests.

NOTE This document uses for hexadecimal values the notation xxh as used in the USB specification. This is different from the notation 'xx' which is used in other parts of this standard series. The notation xxh is used here to avoid possible confusion when reading this document and the related USB documents.

In the following tables of standard descriptors the character asteriks (*) in the column **Value** indicates that this value(s) is defined by ISO/IEC, taken from the set of possible values given in the USB specification. All other values are standard USB entries.

The transmission direction from the host to the USB-ICC is designated as OUT. The transmission direction from the USB-ICC to the host is designated as IN.

7.1.1 The Standard Device Descriptor

Table 1 — Standard device descriptor for a USB-ICC

Offset	Field	Size	Value	Description
0	*bLength*	1	12h	Size of this descriptor in bytes.
1	*bDescriptorType*	1	01h	DEVICE Descriptor Type.
2	*bcdUSB*	2	0200h	USB Specification Release Number.
4	*bDeviceClass*	1	00h*	Indicates that the device class is specified in the interface descriptor of the device.
5	*bDeviceSubClass*	1	00h	Reset to zero as *bDeviceClass* is reset to zero.
6	*bDeviceProtocol*	1	00h*	The device does not use class-specific protocols on the device basis. Instead, it uses class-specific protocols on the interface level.
7	*bMaxPacketSize0*	1		Maximum packet size for endpoint zero. The size may be 8,16,32,64. For low speed functions the value shall be 8.
8	*idVendor*	2		Vendor ID, (assigned by the USB-IF).
10	*idProduct*	2		Product ID, (assigned by the manufacturer). Definition of the value of this field is out of the scope of this document.
12	*bcdDevice*	2		Device release number in binary coded decimal. Definition of the value of this field is out of the scope of this document.
14	*iManufacturer*	1		Index of string descriptor describing manufacturer. Definition of the content of this string is out of the scope of this document.
15	*iProduct*	1		Index of string descriptor describing the product. Definition of the content of this string is out of the scope of this document.
16	*iSerialNumber*	1		Index of string descriptor describing the devices serial number.
17	*bNumConfigurations*	1		Number of possible configurations.

7.1.2 The Standard Configuration Descriptor

Table 2 — Standard configuration descriptor for a USB-ICC

Offset	Field	Size	Value	Description
0	*bLength*	1	09h	Size of this descriptor in bytes.
1	*bDescriptorType*	1	02h	CONFIGURATION Descriptor Type.
2	*wTotalLength*	2		Total length of data returned for this configuration. includes the combined length of all descriptors (configuration, interface, endpoint, and class-specific) returned by this configuration.
4	*bNumInterfaces*	1		The number of interfaces supported by this configuration.
5	*bConfigurationValue*	1		Value to use as an argument to the SetConfiguration() request to select this configuration. This value shall be non-zero.
6	*iConfiguration*	1		Index of string descriptor describing this configuration. Definition of the content of this string is out of the scope of this document.
7	*bmAttributes*	1		Configuration characteristics for the USB-ICC: Bit 4…0: Reserved (reset to zero) Bit 5 Remote Wakeup Bit 6. Self-powered Bit 7 Reserved (set to one) For a bus-powered USB-ICC that does not support remote wake-up, *bmAttributes* shall have the value 80h
8	*MaxPower*	1		Maximum power consumption of the USB-ICC from the bus when the device is fully operational. Expressed in 2mA units.

7.1.3 The Standard Interface Descriptor

Table 3 — Standard interface descriptor for a USB-ICC

Offset	Field	Size	Value	Description
0	*bLength*	1	09h	Size of this descriptor in bytes.
1	*bDescriptorType*	1	04h	INTERFACE Descriptor Type.
2	*bInterfaceNumber*	1		Number of the interface. Zero-based value identifying the index in the array of concurrent interfaces supported by this configuration.
3	*bAlternateSetting*	1	00h*	Value used to select alternate setting for the interface identified in the prior field. Alternate settings are not supported.
4	*bNumEndpoints*	1	00h* 01h* 02h* 03h*	Number of endpoints for a USB-ICC used by this interface (excluding endpoint zero). 00h does not use further endpoints 01h uses interrupt-IN 02h uses bulk-IN and bulk-OUT 03h uses bulk-IN, bulk-OUT and interrupt-IN NOTE 01h indicates that the control endpoints are used for data transmission and interrupt-IN for notification of card specific events sent from the USB-ICC to the host.
5	*bInterfaceClass*	1	0Bh FFh	Class code for the Smart Card device class (0Bh) or the interface class is vendor specific (FFh). NOTE A product not using a class specific driver can be ISO 7816-12 compliant. In this case, the driver will be chosen using the information given by the vendor, the manufacturer and the product ID (see Table 1).
6	*bInterfaceSubClass*	1	00h	Subclass code.
7	*bInterfaceProtocol*	1	00h 01h 02h	Protocol code. The Smart Card device class offers the following interface protocols for a USB-ICC: - 00h USB-ICC messages using bulk (optional interrupt) - 01h USB-ICC specific requests using control transfer Version A (no interrupt) - 02h USB-ICC specific requests using control transfer Version B (optional interrupt) The given value indicates the transfer mode which is used for the communication between host and USB-ICC
8	*iInterface*	1		Index of string descriptor describing this interface. Definition of the content of this string is out of the scope of this document.

7.1.4 The Standard Endpoint Descriptors

A USB-ICC may either communicate with the host using the default control pipe only or it may communicate over message pipes using bulk-IN and bulk-OUT. Optionally, a USB-ICC may provide an interrupt-IN endpoint which allows the USB-ICC to indicate specific events to the host. A USB-ICC may have one of the following configurations:

Table 4 — Configuration of endpoints for a USB-ICC

Endpoints for data transmission	Using control transfers		Using bulk transfers
	Version A	Version B	
Default control pipe	yes	yes	yes
Bulk-IN	no	no	yes
Bulk-OUT	no	no	yes
Interrupt-IN	no	optional	optional

The following tables describe the endpoint descriptors:

Table 5 — Endpoint descriptor bulk-OUT

Offset	Field	Size	Value	Description
0	*bLength*	1	07h	Size of this descriptor in bytes.
1	*bDescriptorType*	1	05h	ENDPOINT descriptor type.
2	*bEndpointAddress*	1	01-0Fh	The address of this endpoint on the USB-ICC. This address is an endpoint number between 1 and 15. Bit 3…0 Endpoint number Bit 6…4 Reserved, must be 0 Bit 7 0 = OUT
3	*bmAttributes*	1	02h	This is a bulk endpoint.
4	*wMaxPacketSize*	2	00xxh	Maximum data transfer size. May be 8, 16, 32, 64.
6	*bInterval*	1	00h	Does not apply to bulk endpoints.

Table 6 — Endpoint descriptor bulk-IN

Offset	Field	Size	Value	Description
0	*bLength*	1	07h	Size of this descriptor in bytes.
1	*bDescriptorType*	1	05h	ENDPOINT descriptor type.
2	*bEndpointAddress*	1	81-8Fh	The address of this endpoint on the USB-ICC. This address is an endpoint number between 1 and 15. Bit 3…0 Endpoint number Bit 6…4 Reserved, must be 0 Bit 7 1 = IN
3	*bmAttributes*	1	02h	This is a bulk endpoint.
4	*wMaxPacketSize*	2	00xxh	Maximum data transfer size. May be 8, 16, 32, 64.
6	*bInterval*	1	00h	Does not apply to bulk endpoints.

Table 7 — Endpoint descriptor interrupt-IN

Offset	Field	Size	Value	Description
0	*bLength*	1	07h	Size of this descriptor in bytes.
1	*bDescriptorType*	1	05h	ENDPOINT descriptor type.
2	*bEndpointAddress*	1	81-8Fh	The address of this endpoint on the USB-ICC. This address is an endpoint number between 1 and 15. It shall be different from the bulk-IN endpoint address. Bit 3…0 Endpoint number Bit 6…4 Reserved, must be 0 Bit 7 1 = IN
3	*bmAttributes*	1	03h	This is an interrupt endpoint.
4	*wMaxPacketSize*	2	00xyh	Packet size for USB-ICC. The minimum value shall be 02h.
6	*bInterval*	1	xyh	Interval for polling endpoint data transfers. Expressed in milliseconds. The value shall be in the range from 1 to 255. In order to save bandwidth, the recommended value is 255.

7.2 The Class Specific Descriptor

The Smart Card device class uses the class specific descriptor as described in the CCID specification (see Annex D (informative)). In the context of a chip card interface device, a USB-ICC represents a configuration of a single slot interface device with a permanently inserted card. The possible values for the class specific descriptor reflect this device configuration. Fields containing *bReserved* or *dwReserved* signify parameters that are not relevant for a USB-ICC. Although not relevant, it is mandatory that a USB-ICC uses exactly these values for *bReserved* and *dwReserved* in order to maintain compatibility with the CCID specification.

Table 8 — Class specific descriptor for a USB-ICC

Offset	Field	Size	Value	Description
0	*bLength*	1	36h	Size of this descriptor, in bytes.
1	*bDescriptorType*	1	21h	CCID Functional Descriptor type.
2	*bcdCCID*	2		CCID Specification Release Number in binary coded decimal. The current version 1.0 is 0100h. CCID Specification Release Number 1.0 will be updated by the USB-DWG Smart Card.
4	*bMaxSlotIndex*	1	00h	Index of the highest available slot. A USB-ICC is regarded as single slot.
5	*bReserved*	1	01h	This value shall be 01h.
6	*dwProtocols*	4	0000 0001h 0000 0002h	Indicates the supported protocol types: 00000001h = Protocol T=0 00000002h = Protocol T=1 NOTE The USB-ICC supports APDU level exchanges for T=1 or character level exchanges for T=0. Other combinations of *dwProtocols* and *dwFeatures* are not supported by the USB-ICC. This applies for bulk transfer mode and for control transfer mode.

Offset	Field	Size	Value	Description
10	*dwReserved*	4	0000 0DFCh	This value shall be 0000 0DFCh.
14	*dwReserved*	4	0000 0DFCh	This value shall be 0000 0DFCh.
18	*bReserved*	1	00h	This value shall be 00h.
19	*dwReserved*	4	0000 2580h	This value shall be 0000 2580h.
23	*dwReserved*	4	0000 2580h	This value shall be 0000 2580h.
27	*bReserved*	1	00h	This value shall be 00h.
28	*dwMaxIFSD*	4		Indicates the maximum IFSD supported by the USB-ICC for protocol T=1. For T=0 any value may be given. For T=1: 000000FEh For T=0: any value
32	*dwReserved*	4	0000 0000h	This value shall be 0000 0000h
36	*dwMechanical*	4	0000 0000h	Indicates that a USB-ICC has no special characteristics.
40	*dwFeatures*	4	0000 0840h 0002 0840h 0004 0840h	The value of the lower word (=0840) indicates that the host will only send requests that are valid for the USB-ICC. The value of the upper word is the level of data exchange with the USB-ICC: 0000h Character level exchanges 0002h Short APDU level exchanges 0004h Short and extended APDU level exchanges NOTE see also *dwProtocols*
44	*dwMaxCCIDMessageLength*	4		For bulk transfers, the value shall be between: 261 + 10 and 65544 +10. NOTE The value 10 is the size of the header For control transfers, the vlue shall be between: 261 and 65544.
48	*bReserved*	1	FFh	This value shall be FFh.
49	*bReserved*	1	FFh	This value shall be FFh.
50	*wRFU*	2	0000h	All other values are reserved for future use
52	*bRFU*	1	00h	All other values are reserved for future use.
53	*bMaxCCIDBusySlots*	1	01h	The USB-ICC is regarded as a single slot.

8 Data transfer between host and USB-ICC

The exchange of data between host and USB-ICC may be done using bulk transfers or control transfers. For control transfer, two implementations are possible. They are named Version A and Version B. Bulk transfer mode is compliant to the CCID specification, e.g. it uses a subset of the messages/requests as defined in this specification.

The notation for the state diagrams is given in the (informative) Annex A.

8.1 Bulk transfers

To transmit commands, responses and corresponding data between host and USB-ICC, the following messages shall apply:

Table 9 — Bulk-IN and bulk-OUT messages

Bulk-OUT message name	Bulk-IN response message name	Description
PC_to_RDR_IccPowerOn	RDR_to_PC_DataBlock	Exits the initial state of a USB-ICC and returns the ATR in the response message.
PC_to_RDR_IccPowerOff	RDR_to_PC_SlotStatus	Sets the USB-ICC to initial conditions.
PC_to_RDR_XfrBlock	RDR_to_PC_DataBlock	Messages to transmit data between host and USB-ICC.

8.1.1 Bulk messages

All messages transmitted over bulk endpoints start with a 10 byte header, optionally followed by data.

The purpose of the header is to exchange control and status information between host and USB-ICC. In addition, sequence numbering assigns command messages with their corresponding response messages. The USB-ICC returns its status and error information in the fields *bStatus* and *bError*.

8.1.1.1 PC_to_RDR_IccPowerOn and RDR_to_PC_DataBlock

Table 10 — PC_to_RDR_IccPowerOn message

Offset	Field	Size	Value	Description
0	*bMessageType*	1	62h	Indicates PC_to_RDR_IccPowerOn.
1	*dwLength*	4	00000000h	There are no extra bytes of this message.
5	*bSlot*	1	00h	Slot number for the USB-ICC.
6	*bSeq*	1	00h – FFh	Sequence number for the command.
7	*bReserved*	1	01h	This value shall be 01h.
8	*abRFU*	2	0000h	All other values are reserved for future use.

The response to this message is the RDR_to_PC_DataBlock message.

Table 11 — RDR_to_PC_DataBlock message containing the ATR

Offset	Field	Size	Value	Description
0	*bMessageType*	1	80h	Indicates RDR_to_PC_DataBlock.
1	*dwLength*	4		Size of bytes for the ATR.
5	*bSlot*	1	00h	Slot number for a USB-ICC.
6	*bSeq*	1	Same value as corresponding bulk-OUT message	Sequence number for the corresponding command message.
7	*bStatus*	1		USB-ICC status information.
8	*bError*	1		Error code in case of failure.
9	*bChainParameter*	1	00h	Indicates that this message contains the complete ATR.
10	*abData*			ATR.

8.1.1.2 PC_to_RDR_IccPowerOff and RDR_to_PC_SlotStatus

Table 12 — PC_to_RDR_IccPowerOff message

Offset	Field	Size	Value	Description
0	*bMessageType*	1	63h	Indicates PC_to_RDR_IccPowerOff.
1	*dwLength*	4	00000000h	There are no extra bytes of this message.
5	*bSlot*	1	00h	Slot number for a USB-ICC.
6	*bSeq*	1	00h – FFh	Sequence number for command.
7	*abRFU*	3	000000h	All other values are reserved for future use.

The response to this message is the RDR_to_PC_SlotStatus message.

Table 13 — RDR_to_PC_SlotStatus message

Offset	Field	Size	Value	Description
0	*bMessageType*	1	81h	Indicates RDR_to_PC_SlotStatus.
1	*dwLength*	4	00000000h	There are no extra bytes of this message.
5	*bSlot*	1	00h	Slot number for a USB-ICC.
6	*bSeq*	1	Same value as corresponding bulk-OUT message	Sequence number for the corresponding command message.
7	*bStatus*	1		USB-ICC status information.
8	*bError*	1		Error code in case of failure.
9	*bReserved*	1	00h	This value shall be 00h

8.1.1.3 PC_to_RDR_XfrBlock and RDR_to_PC_DataBlock

The PC_to_RDR_XfrBlock command is used to transmit command APDUs.

Table 14 — PC_to_RDR_XfrBlock message

Offset	Field	Size	Value	Description
0	*bMessageType*	1	6Fh	Indicates PC_to_RDR_XfrBlock message.
1	*dwLength*	4		Size of *abData* field of this message.
5	*bSlot*	1	00h	Slot number for a USB-ICC.
6	*bSeq*	1	00h – FFh	Sequence number for command.
7	*bReserved*	1	00h	Shall be set to 00h
8	*wLevelParameter*	2		Depends on the exchange level reported by the class specific descriptor in the *dwFeatures* field: - Character level: size of expected data to be returned by the bulk-IN endpoint, - Short APDU level: 00h - Extended APDU level: Indicates if APDU begins or ends in this command: 0000h: the command APDU begins and ends with this command, 0001h: the command APDU begins with this command, and continues in next PC_to_RDR_XfrBlock, 0002h: this *abData* field continues a command APDU and ends the command APDU, 0003h: the *abData* field continues a command APDU and another block is to follow, 0010h: empty *abData* field, continuation of response APDU is expected in the next RDR_to_PC_DataBlock
10	*abData*			Data block sent from host to the USB-ICC.

The response to this message is the RDR_to_PC_DataBlock message.

Table 15 — RDR_to_PC_DataBlock message containing a data block

Offset	Field	Size	Value	Description
0	*bMessageType*	1	80h	Indicates RDR_to_PC_DataBlock.
1	*dwLength*	4		Size of bytes for the received data block.
5	*bSlot*	1	00h	Slot number for a USB-ICC.
6	*bSeq*	1	Same value as bulk-OUT message	Sequence number for the corresponding command message.

Offset	Field	Size	Value	Description
7	*bStatus*	1		USB-ICC status information.
8	*bError*	1		Error code in case of failure.
9	*bChainParameter*	1		Depends on the exchange level reported by the class specific descriptor in the *dwFeatures* field: - Character level: 00h - Short APDU: 00h - Extended APDU level: Indicates if the response is complete, to be continued or if the command APDU can continue: 00h: the response APDU begins and ends in this command, 01h: the response APDU begins with this command and is to continue, 02h: this abData field continues the response APDU and ends the response APDU, 03h: this abData field continues the response APDU and another block is to follow, 10h: empty *abData* field, continuation of the command APDU is expected in the next PC_to_RDR_XfrBlock command.
10	*abData*			Data sent from USB-ICC to the host.

8.1.2 ATR and transmission of data

When a USB device is attached to the bus and thereafter has obtained a state where the host may use the functions provided by the device, the device is designated as "Configured". The messages to be transmitted in order to set the USB-ICC to initial state, to obtain the ATR and to transmit data are given in the state diagram Figure 2. The transmission uses APDU level exchanges. Figure 2 comprises the transmission of short APDUs and extended APDUs.

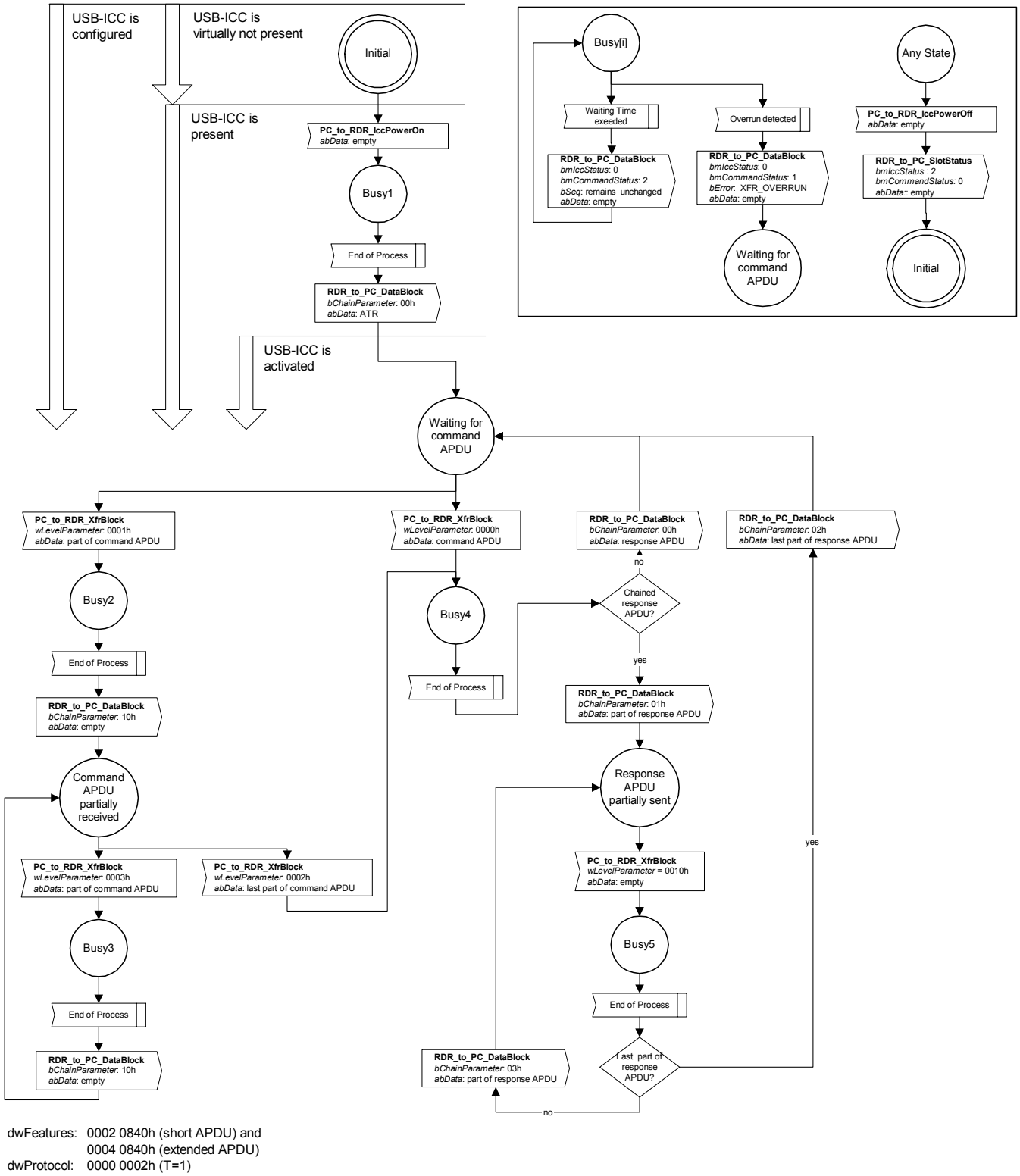
For the correct transmission of data, the following general rules shall apply:

— If the USB-ICC receives a PC_to_RDR_PowerOn when it is not in the state "Initial", the USB-ICC shall respond with a STALL. The USB-ICC shall remain in its current state.

— If the USB-ICC requests a time extention (see Table 16), the value of *bSeq* (see clause 8.1.1) shall remain unchanged.

— If the USB-ICC returns RDR_to_PC_DataBlock indicating the errors ICC_MUTE or HW_ERROR, the host should preferably submit a PC_to_RDR_IccPowerOff message.

IMPORTANT — The state of the current execution shall not be affected by the state of the USB interface engine. For example, a bus enumeration shall not cause any transitions.



dwFeatures: 0002 0840h (short APDU) and
0004 0840h (extended APDU)
dwProtocol: 0000 0002h (T=1)

Figure 2 — State diagram of the USB-ICC using bulk transfer APDU level transfer for short APDU and extended APDU

8.1.3 Status and error conditions

The bulk-IN messages RDR_to_PC_SlotStatus and RDR_to_PC_DataBlock contain status information about the USB-ICC and if the processed commands completed successfully. In case of a failure, an error code will be returned.

The *bStatus* field consists of two bitmap fields that contain information about the USB-ICC status (*bmICCStatus*) and the processed command (*bmCommandStatus*). The following two tables give the values for the status and the error codes.

Table 16 — Bitmap for bStatus field

Offset	Field	Size	Value	Description
0	*bmIccStatus*	1 (2 bits)	0, 1, 2	0 = The USB-ICC is present and activated. 1 = The USB-ICC is present but not activated 2 = The USB-ICC is virtually not present 3 = RFU
(2 bits)		(4 bits)		RFU
(6 bits)	*bmCommandStatus*	(2 bits)	0, 1, 2	0 = Processed without error. 1 = Failed, error condition given by *bError*. 2 = Time extention is requested 3 = RFU
1	*bError*	1		Error codes

Table 17 — Error codes for bError

Error name	Error code	Possible causes
ICC_MUTE	-2 (FEh)	The applications of the USB-ICC did not respond or the ATR could not be sent by the USB-ICC.
XFR_OVERRUN	-4 (FCh)	The USB-ICC detected a buffer overflow when receiving a data block.
HW_ERROR	-5 (FBh)	The USB-ICC detected an hardware error.
	-64 to –127 (C0h – 81h)	User defined
	-3 (FDh) -8 to –14 (F8h –F2h) -16 (F0h) -17 (EFh) -32 (E0h)	These values shall not be used by the USB-ICC
	all others (80h and those filling the gaps)	Reserved for future use

For the usage of error codes, the following rules shall apply:

— if the value of *bmCommandStatus* equals 0 or RFU, the value of *bError* shall be 0.

— if the value of *bmCommandStatus* equals 1, the value of *bError* shall be:

  — error code = error conditions as described in Table 17.

  — offset = if the USB-ICC can not parse one field in the (10 byte) header or is not supporting one of these fields, then *bError* contains the offset of the first bad value as a positive number (e.g. if the host sets *bSlot* to 01h, the USB-ICC will return *bError* = 05h). A USB-ICC receiving a command that is not supported, shall set the offset value to zero.

8.2 Control transfers

This transfer mode can be employed for USB-ICCs that offer low speed functions. The default control pipe is used to exchange data between host and USB-ICC.

This paragraph defines the class specific requests for control transfer. These requests provide the same services to the application layer as for bulk transfers.

There are two implementations for control transfers, named hereafter as Version A and Version B.

8.2.1 Version A

8.2.1.1 Specific requests

The following table defines valid values of bRequest*:*

Table 18 — Class specific requests, Version A

bRequest	Value	Direction data stage	Description
ICC_POWER_ON	62h	IN	Exits the initial state of a USB-ICC. Returns the ATR in the data stage.
ICC_POWER_OFF	63h	OUT	Sets the USB-ICC to initial conditions.
XFR_BLOCK	65h	OUT	Data transfer from the host to the USB-ICC
DATA_BLOCK	6Fh	IN	Data transfer from the USB-ICC to the host
GET_ICC_STATUS	A0h	IN	Returns the status of the command execution.

8.2.1.2 Setup Stage

The setup stage contains the class specific request and corresponding parameters. The following tables give the values and the parameters for each of the class specific requests and describe the data that is transferred between host and USB-ICC.

For the parameters, the following general rules shall apply:

— The value of *bInterface* is the same value as *bInterfaceNumber* given in Table 3.

— Reserved parameter values for class specific requests used in the fields wValue and wIndex are designated as bRFU and wRFU. The value of bRFU shall be set to 00h and the value of wRFU shall be set to 0000h.

— If the USB-ICC receives an invalid request or if a valid request contains an invalid parameter value (wValue, wIndex,wLength), the USB-ICC shall respond with a STALL.

— On an input request, the USB-ICC shall not return more data than is indicated by wLength value. It may return less. On an output request, wLength shall always indicate the exact amount of data to be sent by the host. The USB-ICC shall return a STALL if the host should not send the amount of data than is specified in wLength.

— For ICC_POWER_OFF and GET_ICC_STATUS, the host shall send the values for wLength as specified in the corresponding tables. If not, the USB-ICC shall respond with STALL

Table 19 — ICC_POWER_ON request, Version A

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100001B	ICC_POWER_ON	wRFU	bRFU *bInterface*	Length of ATR	ATR

The wIndex field specifies bRFU in the high byte and *bInterface* in the low byte.

Table 20 — ICC_POWER_OFF request, Version A

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	ICC_POWER_OFF	wRFU	bRFU *bInterface*	0000h	Empty

The wIndex field specifies bRFU in the high byte and *bInterface* in the low byte.

Table 21 — XFR_BLOCK request, Version A

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	XFR_BLOCK	*bLevelParameter* bRFU	bRFU *bInterface*	Length of data	Command APDU.

The wIndex field specifies bRFU in the high byte and *bInterface* in the low byte. The wValue field specifies *bLevelParameter* in the high byte and bRFU in the low byte.

Table 22 — DATA_BLOCK request, Version A

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100001B	DATA_BLOCK	wRFU	bRFU *bInterface*	Length of data	Response APDU

The wIndex field specifies bRFU in the high byte and *bInterface* in the low byte.

Table 23 — GET_ICC_STATUS request

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100001B	GET_ICC_STATUS	wRFU	bRFU *bInterface*	0001h	Polls the status of the USB-ICC

The wIndex field specifies bRFU in the high byte and *bInterface* in the low byte.

NOTE A product not using a class specific driver can be 7816-12 compliant. In this case, the coding of bit 5,6 of bmRequestType changes from 01B to 10B; for example bmRequestType for GET_ICC_STATUS will be 11000001B

8.2.1.3 ATR and data transmission

When a USB device is attached to the bus and thereafter has obtained a state where the host may use the functions provided by the device, the device is designated as "Configured". The messages to be transmitted in order to set the USB-ICC to initial state, to obtain the ATR and to transmit data are given in the following state diagrams (Figure 3,4,5).

For the correct transmission of data, the following general rules shall apply:

— If the USB-ICC receives a request that is not assigned to the current state as defined in the state diagram, the USB-ICC shall return a STALL and remain in its current state.

— If the StatusByte indicates that the card is not responsive (see Table 24), the host should preferably submit ICC_POWER_OFF.

IMPORTANT — The state of the current execution shall not be affected by the state of the USB interface engine. For example, a bus enumeration shall not cause any transitions.

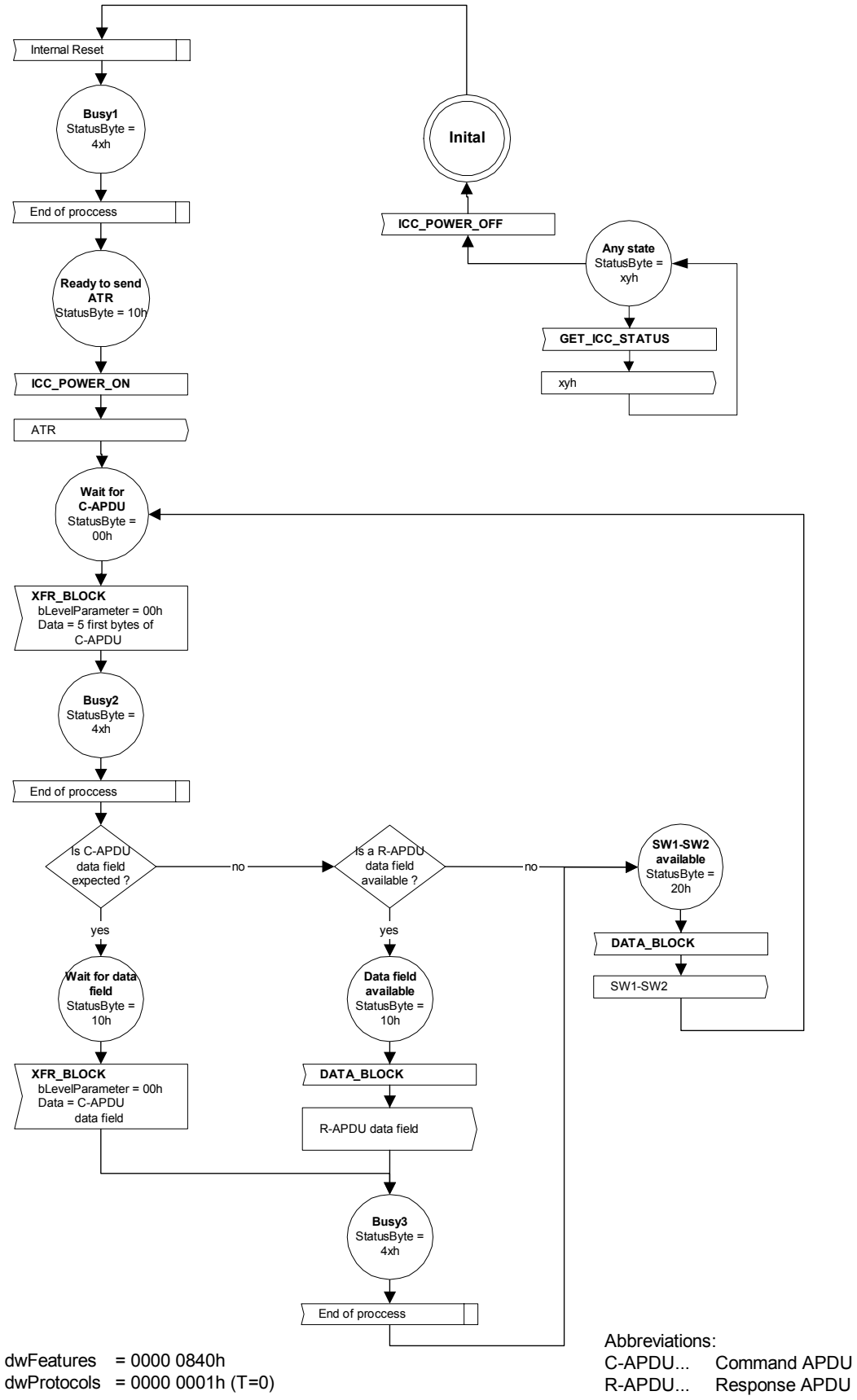
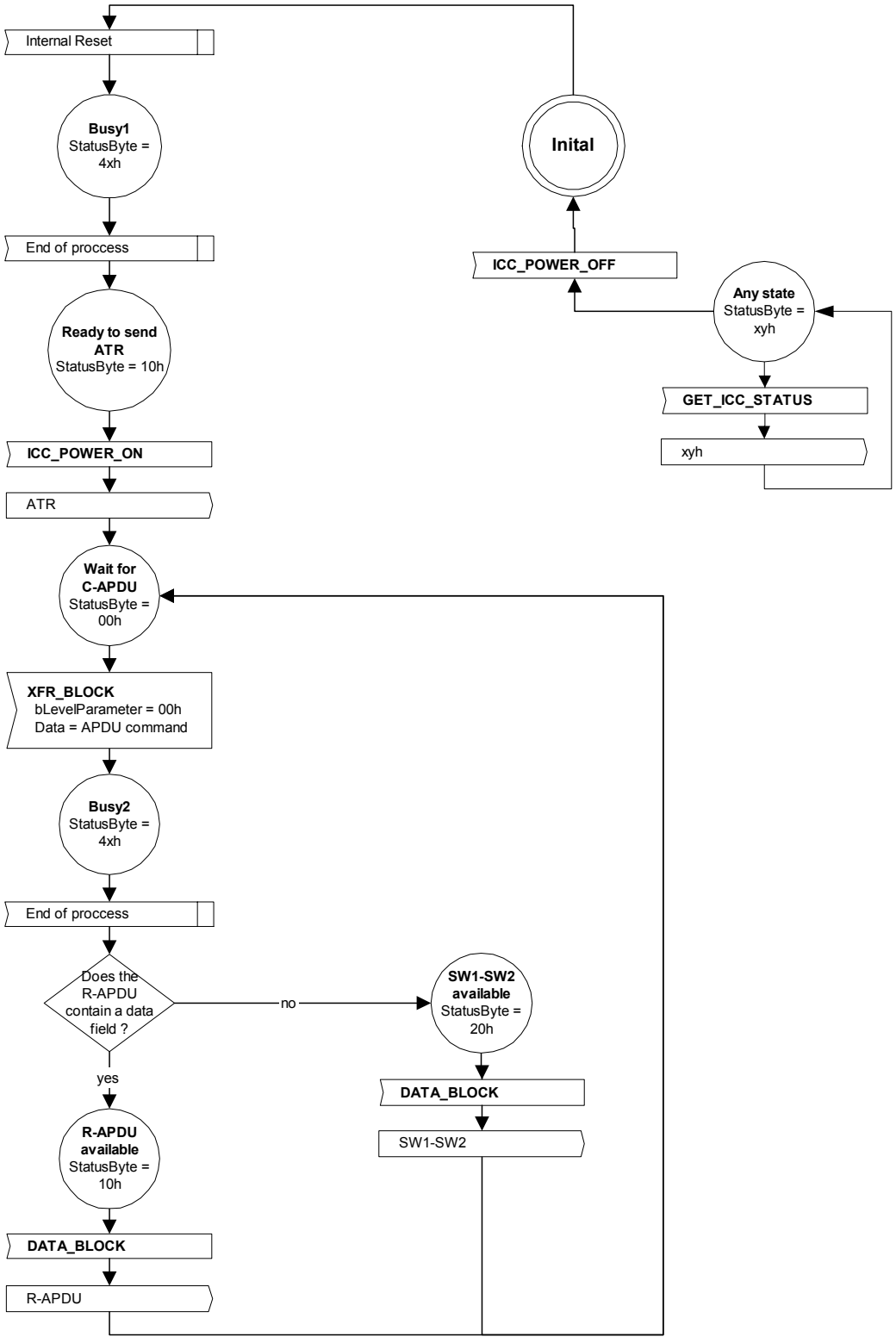


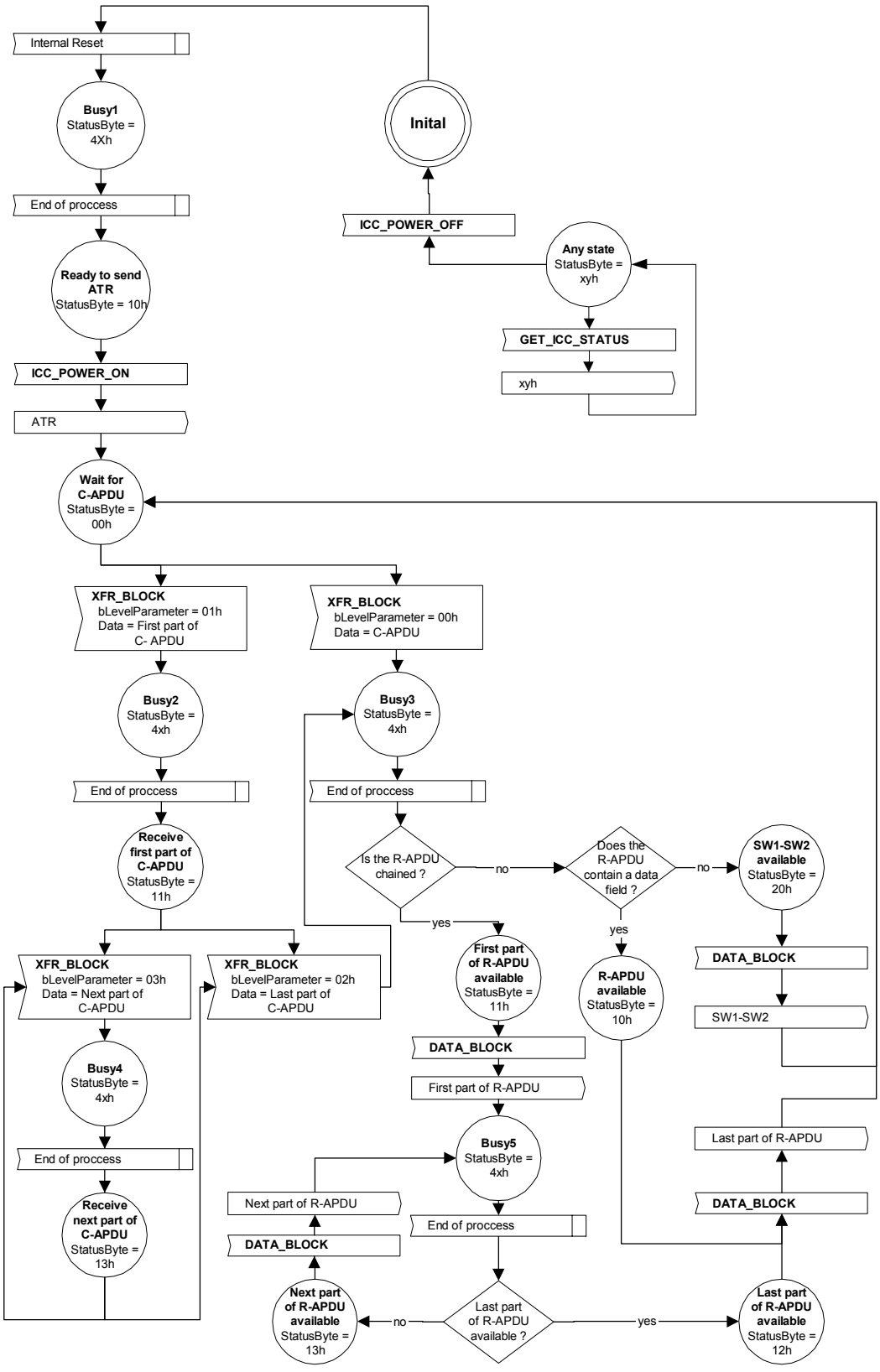
Figure 3 — State diagram of the USB-ICC for control transfer (Version A) using character level transfer



dwFeatures = 0002 0840h
dwProtocols = 0000 0002h (T=1)

Abbreviations:
C-APDU... Command APDU
R-APDU... Response APDU

Figure 4 — State diagram of the USB-ICC for control transfer (Version A) using short APDUs



dwFeatures = 0004 0840h
dwProtocols = 0000 0002h

Abbreviations:
C-APDU... Command APDU
R-APDU... Response APDU

Figure 5 — State diagram of the USB-ICC for control transfer (Version A) using extended APDUs

The request GET_ICC_STATUS polls the status of execution of a command APDU. Upon this request, the USB-ICC returns the StatusByte to indicate the status of execution. It may have the following values:

Table 24 — Description of the StatusByte

StatusByte	Description
4xh	**busy** where x shall be cyclically incremented. When receiving a **busy** indication, the host shall subsequently submit GET_ICC_STATUS until the USB-ICC indicates another value. The time interval is driver dependent. In order to save bandwidth, the time interval should not be less than 10ms. NOTE When the host detects that the four least significant bits did not change after a certain period, the host might time-out the device. The period, which is considered as time-out, is driver dependent and should not be less than 1 second.
20h	**ready to send status words only** Indicates that the data stage of the subsequent DATA_BLOCK will convey SW1-SW2 only.
1yh	if *dwProtocols*=00000001h and *dwFeatures*=00000840h 10h: **ready to send data** or 10h: **ready to receive data** The status words are not returned when the value is 10. When GET_ICC_STATUS returns StatusByte=20h, a subsequent DATA_BLOCK request shall be submitted to obtain the status words. if *dwProtocols*=00000002h and *dwFeatures*=000z0840h (with z=2 or z=4) the StatusByte has two different functions. When *bLevelParameter* is 01h or 03h in the previous XFR_BLOCK request (chained command APDU), the StatusByte is used to acknowledge the chaining of the command (respectively 11h or 13h) and to regulate the data flow (StatusByte = 4xh). When *bLevelParameter* in the previous XFR_BLOCK request is 00h or 02h (end of command APDU), the StatusByte is used to indicate the chaining of the response APDU and to regulate the data flow (StatusByte = 4xh): 10h the APDU response begins and ends with the next DATA_BLOCK request 11h the APDU response begins with the next DATA_BLOCK request and is to continue 12h the APDU response continues and ends with the next DATA_BLOCK request 13h the APDU response continues with the next DATA_BLOCK request and another block is to follow 20h the APDU response contains only the status word and ends with the next DATA_BLOCK request.
80h	**mute** the card is not responsive
00h	The USB-ICC is ready to receive a command APDU

8.2.1.4 APDU level message exchange

In case that the length of the response APDU exceeds the value of wLength in the status stage of the DATA_BLOCK request, the response APDU has to be transmitted in subsequent blocks. In this case, the USB-ICC shall use the same mechanism as for an extended response APDU.

8.2.1.5 Error conditions

Error conditions are returned in the StatusByte. If the card is not responsive, the value 80h will be returned.

8.2.1.6 Interrupt Transfers

Version A does not use interrupt transfers.

8.2.2 Version B

Control requests under Version B are similar to the exchange of information when using message pipes in bulk mode. This is achieved by the fact that each OUT-requests are to be followed by IN-requests. This pair wise use of requests reflects the structure of OUT-messages and IN-messages for bulk mode.

8.2.2.1 Specific requests

The following table defines valid values of bRequest*:*

Table 25 — Class specific requests, Version B

bRequest	Value	Direction data stage	Description
ICC_POWER_ON	62h	OUT	Exits the initial state of a USB-ICC. The ATR is returned in the data stage of the subsequent DATA_BLOCK request.
ICC_POWER_OFF	63h	OUT	Sets the USB-ICC to initial conditions.
XFR_BLOCK	65h	OUT	Data transferred from the host to the USB-ICC
DATA_BLOCK	6Fh	IN	Data transferred from the USB-ICC to the host. Also returns information created by the preceding request.
SLOT_STATUS	81h	IN	The data stage of this command contains *bStatus, bError* and *bReserved*. The value for *bReserved* shall be 00h.

8.2.2.2 Setup Stage

The setup stage contains the class specific request and corresponding parameters. The following clauses give the values and the parameters for each of the class specific requests and describe the data that is transferred between host and USB-ICC.

The parameters of the class specific request shall be set as follows:

— The value of *bInterface* is the same value as *bInterfaceNumber* given in Table 3.

— Reserved parameter values for class specific requests used in the fields wValue and wIndex are designated as bRFU, wRFU and *bReserved*. The value of bRFU shall be set to 00h and the value of wRFU shall be set to 0000h. The value for *bReserved* is given in the tables.

— If the USB-ICC receives an invalid request or if a valid request contains an invalid parameter value (wValue, wIndex,wLength), the USB-ICC shall respond with a STALL.

— On an input request, the USB-ICC shall not return more data than is indicated by wLength value. It may return less.

— On an output request, wLength shall always indicate the exact amount of data to be sent from the host to the USB-ICC. When the USB-ICC receives more data as indicated in the setup stage, it shall respond with STALL. The host may abort any transfer by sending an IN-token prematurely. In this case the USB-ICC shall confirm the IN-token with ACK. If the number of received data is not equal to wLength, the USB-ICC shall discard these data.

— For ICC_POWER_ON, ICC_POWER_OFF and SLOT_STATUS, the host shall send the values for wLength as specified in the corresponding tables. If not, the USB-ICC shall respond with STALL

Table 26 — ICC_POWER_ON request, Version B

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	ICC_POWER_ON	bRFU *bReserved*=01h	bRFU *bInterface*	0000h	Empty

The wIndex field specifies bRFU in the high byte and *bInterface* in the low byte. The wValue field specifies *bRFU* in the high byte and *bReserved* in the low byte.

Table 27 — ICC_POWER_OFF request, Version B

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	ICC_POWER_OFF	wRFU	bRFU *bInterface*	0000h	Empty

The wIndex field specifies bRFU in the high byte and *bInterface* in the low byte.

Table 28 — XFR_BLOCK request, Version B

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	XFR_BLOCK	*bLevelParameterb Reserved*=00h	bRFU *bInterface*	Length data	Command APDU

The wIndex field specifies bRFU in the high byte and *bInterface* in the low byte. The wValue field specifies *bLevelParameter* in the high byte and *bReserved* in the low byte.

The use of *bLevelParameter* is explained in Table 14.

Table 29 — DATA_BLOCK request, Version B

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100001B	DATA_BLOCK	wRFU	bRFU *bInterface*	Length of data + 1 The value of wLength shall be greater than or equal to 4. This allows the USB-ICC to return at minimum the complete status information (see also Table 31).	Response APDU or information created by ICC_POWER_ON. NOTE The DATA_BLOCK request always returns in its data stage first one byte (*bResponseType*, see Table 31) followed by the data being processed.

The wIndex field specifies bRFU in the high byte and *bInterface* in the low byte.

Table 30 — SLOT_STATUS request

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100001B	SLOT_STATUS	wRFU	bRFU *bInterface*	0003h	Contains status/error information: *bStatus, bError, bReserved* The value for *bReserved* shall be 00h

The wIndex field specifies bRFU in the high byte and *bInterface* in the low byte.

NOTE A product not using a class specific driver can be 7816-12 compliant. In this case, the coding of bit 5,6 of bmRequestType changes from 01B to 10B; for example bmRequestType for SLOT_STATUS will be 11000001B

8.2.2.3 ATR and data transmission

When a USB device is attached to the bus and thereafter has obtained a state where the host may use the functions provided by the device, the device is designated as "Configured". The messages to be transmitted in order to set the USB-ICC to initial state, to obtain the ATR and to transmit data are given in the state diagram Figure 6. The transmission uses APDU level exchanges. Figure 6 comprises the transmission of short APDUs and extended APDUs.

For the correct transmission of data, the following general rules shall apply to the state diagram:

— If the USB-ICC receives a request that is not assigned to the current state as defined in the state diagram, the USB-ICC shall return a STALL and remain in its current state.

— If the interface device sends DATA_BLOCK and the USB-ICC returns in the data stage the errors ICC_MUTE or HW_ERROR, the host should preferably submit ICC_POWER_OFF.

IMPORTANT — The state of the current execution shall not be affected by the state of the USB interface engine. For example, a bus enumeration shall not cause any transitions.

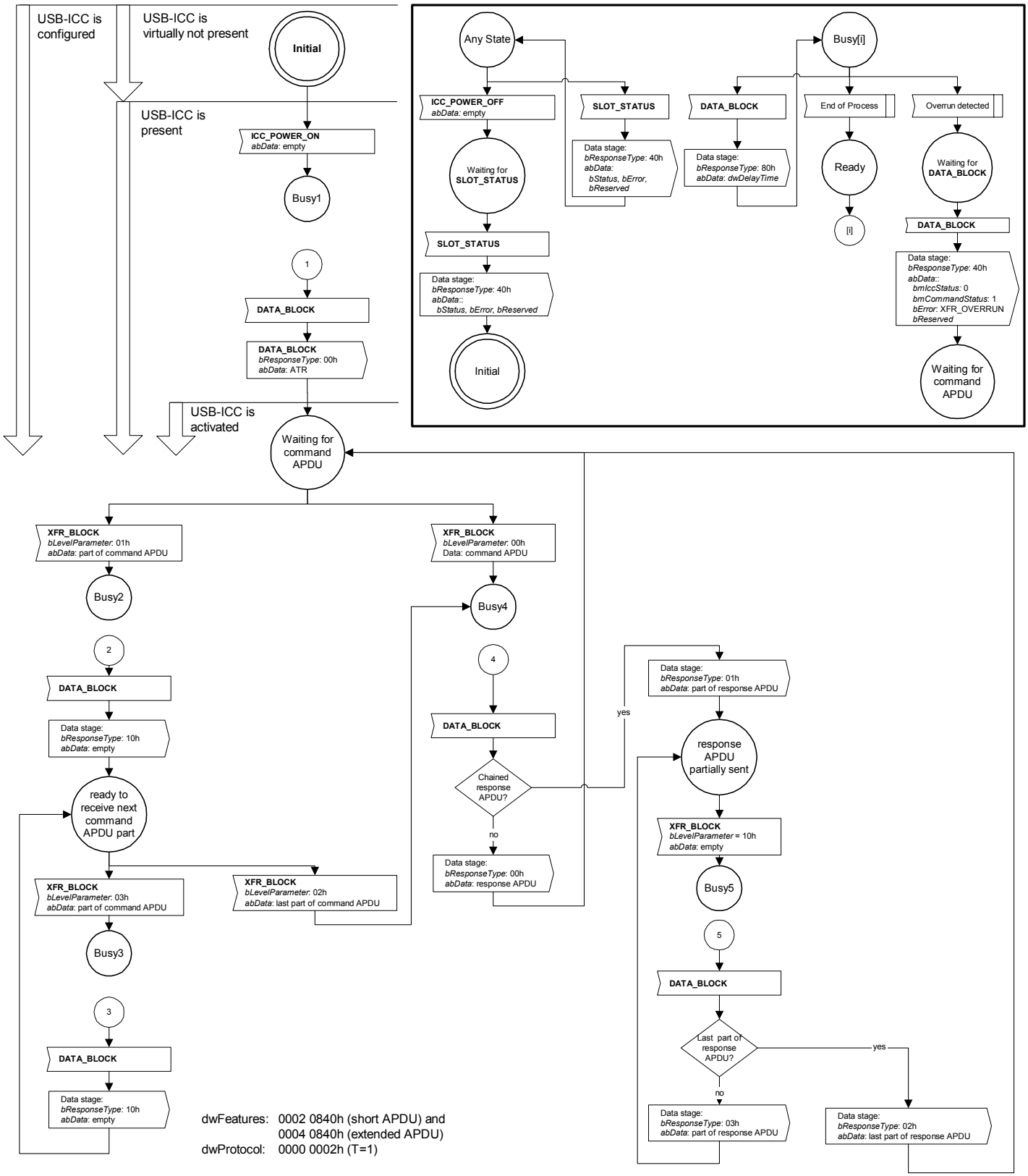


Figure 6 — State diagram of the USB-ICC for control transfer (Version B)

NOTE When the host has sent ICC_POWER_OFF, the USB-ICC enters the state "virtually not present". The use of interrupt-IN messages for this case is described in clause 9.3

The USB-ICC returns on the DATA_BLOCK request the following values in the data stage:

Table 31 — Data stage of DATA_BLOCK

Offset	Field	Description
0	*bResponseType*	Indicates the type of information *abData* field contain: 00h: the *abData* field contains the information created by the preceding request. 40h: Status information the *abData* field contains *bStatus, bError* and *bReserved*=00h. 80h: Polling the *abData* field contains the delay time (*wDelayTime*) until the host waits to send out the next request. The value is given in units of 10ms (e.g. 0078h = 1,2s). If *wDelayTime* equals 0000h, the host shall set the polling interval at its own discretion. For all other values, the host shall use the given value at best effort. For extended response APDUs: 00h: the response APDU begins and ends in this command, 01h: the response APDU begins with this command and is to continue, 02h: this *abData* field continues the response APDU and ends the response APDU, 03h: this *abData* field continues the response APDU and another block is to follow. 10h: empty *abData* field, continuation of the command APDU is expected in the next XFR_BLOCK. See also the state diagram in Figure 6.
1	*abData*	Data sent from the USB-ICC to the host

The information that is transmitted in *abData* field of DATA_BLOCK, depends on the preceding request.

8.2.2.4 Coding of *bLevelParameter* for XFR_BLOCK

The value of *bLevelParameter* designates the position (first block, middle, last) of the subsequently transmitted blocks of a command APDU. The following values are assigned:

00h the command APDU begins and ends with this command
01h the command APDU begins with this command and is to continue
02h the command APDU continues and ends the command APDU
03h the command APDU continues and another block is to follow
10h the data stage is empty, continuation of response APDU is expected in the next DATA_BOCK request

See also the state diagram in Figure 6.

8.2.2.5 APDU level message exchange

In case that the length of the response APDU exceeds the value of wLength in the setup stage of the DATA_BLOCK request, the response APDU has to be transmitted in subsequent blocks. For this block wise transmission, the USB-ICC shall use the same mechanism as for extended response APDUs.

8.2.2.6 Status and error conditions reported by USB requests

Version B returns status and error conditions in the data stage of DATA_BLOCK. This condition is indicated by *bResponseType*=40h. In addition, the USB-ICC will respond with a STALL handshake when it receives an invalid request or if a valid request contains an invalid parameter value (wValue, wIndex, wLength).

If *bResponseType*=40h, *abData* field contains status and error information.

The *bStatus* field consists of two bitmap fields that contain information about the USB-ICC status (*bmICCStatus*) and the processed command (*bmCommandStatus*). The following two tables give the values for the status and the error codes.

Table 32 — Bitmap for bStatus field

Offset	Field	Size	Value	Description
0	*bmIccStatus*	1 (2 bits)	0, 1, 2	0 = The USB-ICC is present and activated. 1 = The USB-ICC is present but not activated 2 = The USB-ICC is virtually not present 3 = RFU
(2 bits)		(4 bits)		RFU
(6 bits)	*bmCommandStatus*	(2 bits)	0, 1	0 = Processed without error. 1 = Failed, error condition given by *bError*. 2 = RFU 3 = RFU
1	*bError*	1		Error codes

Table 33 — Error codes for bError

Error name	Error code	Possible causes
ICC_MUTE	-2 (FEh)	The applications of the USB-ICC did not respond or the ATR could not be sent by the USB-ICC.
XFR_OVERRUN	-4 (FCh)	The USB-ICC detected a buffer overflow when receiving a data block.
HW_ERROR	-5 (FBh)	The USB-ICC detected an hardware error.
	-64 to -127 (C0h – 81h)	User defined
	-3 (FDh) -8 to –14 (F8h –F2h) -16 (F0h) -17 (EFh) -32 (E0h)	These values shall not be used by the USB-ICC
	all others (80h and those filling the gaps)	Reserved for future use

If the value of *bmCommandStatus* equals 0 or RFU, the value of *bError* shall be 0.

8.3 Interrupt transfers

Bulk transfer mode and control transfer mode (Version B) optionally provide an interrupt-IN endpoint. This endpoint is used to notify the host of events that may occur asynchronously to the command/response exchange between host and USB-ICC.

The USB-ICC may notify to the host its virtual insertion/removal.

8.3.1 Virtual insertion/removal event

The term "virtual" is used to express that the USB-ICC may be envisaged as removed from the interface device although it is still powered.

Table 34 — Interrupt-IN message

Offset	Field	Size	Value	Description
0	*bMessageType*	1	50h	Indicates NotifySlotChange.
1	*bmSlotIccState*	1	000000xyB	The USB-ICC indicates the status (removed or inserted) in two least significant bits. The least significant bit reports: 0b= TheUSB-ICC is virtually not present, 1b= The USB-ICC is present. The other bit reports whether the virtual presence of the USB-ICC has changed since the last NotifySlotChange message was sent: 0b= no change 1b= change All other bits shall be set to zero.

For correct operation of the interrupt request, the following conditions shall apply:

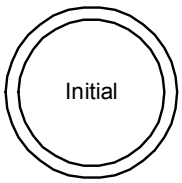
— When the USB-ICC exits from the state "Initial" by PC_to_RDR_IccPowerOn (ICC_POWER_ON), the USB-ICC shall send a NotifySlotChange message with *bmSlotIccState*=00000011B.

— The USB-ICC may enter “virtually not present” at any point in time. The host will receive the NotifySlotChange message with *bmSlotIccState*=00000010B. The USB-ICC shall not send the NotifySlotChange message after it has received a PC_to_RDR_IccPowerOff (ICC_POWER_OFF).

NOTE The first condition allows the host to detect an unresponsive card. The second condition ensures that the interrupt message "virtually not present" is an asynchronous event caused by the USB-ICC. It is not the result of an OUT message or request received from the host.

Annex A
(informative)

Notation for the state diagrams

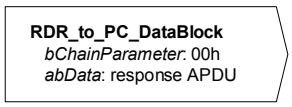


Initial state in the state diagram

State in the state diagram

The USB-ICC receives data.

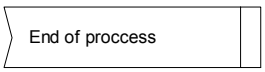
For bulk transfers:
The message type and relevant parameters/values are given.
For control transfers:
The control request and relevant parameters/values are given



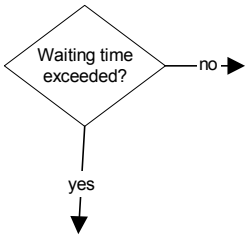
The USB-ICC sends data.

For bulk transfers:
The message type and relevant parameters/values are given. The USB-ICC always initiates the transmission of this message.

For control transfers:
The control request and relevant parameters/values are given. The host always initiates the transmission of this data by sending the DATA_BLOCK or SLOT_STATUS. Therefore, these two requests are always represented by two arrowed boxes: a box with an IN-arrow (request) and a box with an OUT-arrow (data).



The USB-ICC receives data. from an (USB-ICC) internal process



Decision branch

Annex B
(informative)

Scenarios for USB transfers

The sequences described in here are not exhaustive of all possible cases. They are meant as examples of possible sequences the USB-ICC must be able to handle. The examples do neither handle error cases (STALL on USB transactions), nor conditions that are handled on the ISO protocol level (for example T=0, case 2, short APDU, N_e not accepted).

Bulk transfer:
APDU level message exchange, case 3 command, short APDU

PC_to_RDR_XfrBlock
dwLength = Lc + 00000005h
bSlot = 00h, *bSeq* = 00h, *bReserved* = 00h,
wLevelParameter = 0000h
abData = CLA, INS, P1, P2, Lc, Data(Lc)

(1) message →

(2) message ← **RDR_to_PC_DataBlock**
dwLength = 00000002h
bSlot = 00h, *bSeq* = 00h
bStatus = 00h, *bError* = 00h
bChainParameter = 00h
abData = SW1, SW2

Bulk transfer:
APDU level message exchange, case 3 command, extended APDU

PC_to_RDR_XfrBlock
dwLength = BufferLength
bSlot = 00h, *bSeq* = 00h, *bReserved* = 00h,
wLevelParameter = 0001h
abData = CLA, INS, P1, P2, Lc,
Data(BufferLength – 7)

(1) message →

(2) message ← RDR_to_PC_DataBlock
dwLength = 00000000h
bSlot = 00h, *bSeq* = 00h
bStatus = 00h, *bError* = 00h
bChainParameter = 10h
abData = empty

PC_to_RDR_XfrBlock
dwLength = BufferLength
bSlot = 00h, *bSeq* = 01h, *bReserved* = 00h,
wLevelParameter = 0003h
abData = Data(BufferLength)

(3) message →

(4) message ← RDR_to_PC_DataBlock
dwLength = 00000000h
bSlot = 00h, *bSeq* = 01h
bStatus = 00h, *bError* = 00h
bChainParameter = 10h
abData = empty

PC_to_RDR_XfrBlock
dwLength = remaining part of data
bSlot = 00h, *bSeq* = 00h, *bReserved* = 00h,
wLevelParameter = 0002h
abData = Data(<buffer length)

(5) message →

(5) message ← RDR_to_PC_DataBlock
dwLength = 00000002h
bSlot = 00h, *bSeq* = 02h
bStatus = 00h, *bError* = 00h
bChainParameter = 00h
abData = SW1, SW2

Control transfer, Version A: Case 1 command

dwFeatures = 0000 0840h
dwProtocols= 0000 0001h

XFR_BLOCK wValue: *bLevelParameter* = 00h*; bRFU* = 00h wIndex: bRFU = 00h*; bInterface* = 00h wLength = 0005h abData = CLA, INS, P1, P2, P3=00h	**(1) request** →		
	(1) data →		
GET_ICC_STATUS wValue: wRFU = 0000h wIndex: bRFU = 00h*; bInterface* = 00h, wLength = 0001h	**(2) request** →		
	(2) data ←	StatusByte: 4xh StatusByte: 20h	
if (StatusByte=20h)	Break		
if (StatusByte=4xh)	(**2**) Repeat		
	ready to send status words only		
DATA_BLOCK wValue: wRFU = 0000h wIndex: bRFU = 00h*; bInterface* = 00h wLength = 0002h	**(3) request** →		
	(3) data ←	SW1, SW2	
	End		

Control transfer, Version A: Case 2 command, short APDU

dwFeatures = 0000 0840h
dwProtocols= 0000 0001h

XFR_BLOCK wValue:*bLevelParameter* = 00h, *bRFU* = 00h wIndex: bRFU = 00h*, bInterface* = 00h wLength = 0005h abData = CLA, INS, P1, P2, Le	**(1) request** →		
	(1) data →		
GET_ICC_STATUS wValue: wRFU = 0000h wIndex: bRFU = 00h, *bInterface* = 00h wLength = 0001h	**(2) request** →		
	(2) data ←	StatusByte:	4xh
		StatusByte:	10h
if (StatusByte=10h)	Break		
if (StatusByte=4xh)	(**2**) Repeat		
	ready to send data		
DATA_BLOCK wValue: wRFU = 0000h wIndex: bRFU = 00h*, bInterface* = 00h wLength = Le	**(3) request** →		
	(3) data ←	Data(Le)	
GET_ICC_STATUS wValue: wRFU = 0000h wIndex: bRFU = 00h, *bInterface* = 00h wLength = 0001h	**(4) request** →		
	(4) data ←	StatusByte:	4xh
		StatusByte:	20h
if (StatusByte=20h)	Break		
if (StatusByte=4xh)	(**4**) Repeat		
	ready to send status words only		
DATA_BLOCK wValue: wRFU = 0000h wIndex: bRFU = 00h, *bInterface* = 00h wLength = 0002h	**(5) request** →		
	(5) data ←	SW1, SW2	
	End		

Control transfer, Version A: Case 3 command, short APDU

dwFeatures = 0000 0840h
dwProtocols= 0000 0001h

XFR_BLOCK wValue: *bLevelParameter* = 00h, *bRFU* = 00h wIndex: bRFU = 00h, *bInterface* = 00h wLength = 0005h abData = CLA, INS, P1, P2, Lc	**(1) request** →		
	(1) data →		
GET_ICC_STATUS wValue: wRFU = 0000h wIndex: bRFU = 00h, *bInterface* = 00h wLength = 0001h	**(2) request** →		
	(2) data ←	StatusByte:	4xh
		StatusByte:	10h
if (StatusByte=10h)	Break		
if (StatusByte=4xh)	(**2**) Repeat		
	ready to receive data		
XFR_BLOCK wValue: *bLevelParameter* = 00h, *bRFU* = 00h wIndex: bRFU = 00h, *bInterface* = 00h wLength = Lc abData = Data(Lc)	**(3) request** →		
	(3) data →		
GET_ICC_STATUS wValue: wRFU = 0000h wIndex: bRFU = 00h, *bInterface* = 00h wLength = 0001h	**(4) request** →		
	(4) data ←	StatusByte:	4xh
		StatusByte:	20h
if (StatusByte=20h)	Break		
if (StatusByte=4xh)	(**4**) Repeat		
	ready to send status words only		
DATA_BLOCK wValue: wRFU = 0000h wIndex: bRFU = 00h, *bInterface* = 00h wLength = 0002h	**(5) request** →		
	(5) data ←	SW1, SW2	
	End		

Control transfer, Version A:
APDU level message exchange, case 1 command

dwFeatures = 0002 0840h
dwProtocols= 0000 0002h

XFR_BLOCK	**(1) request**	→	
wValue: *bLevelParameter* = 00h, bRFU = 00h wIndex: bRFU = 00h, *bInterface* = 00h wLength = 0004h abData = CLA, INS, P1, P2	(1) data	→	
GET_ICC_STATUS	**(2) request**	→	
wValue: wRFU = 0000h wIndex: bRFU = 00h, *bInterface* = 00h wLength = 0001h	(2) data	←	StatusByte: 4xh StatusByte: 20h
if (StatusByte=20h)	Break		
if (StatusByte=4xh)	(**2**) Repeat		
DATA_BLOCK	**(3) request**	→	
wValue: wRFU = 0000h wIndex: bRFU = 00h, *bInterface* = 00h wLength = 0002h	(3) data	←	SW1, SW2
	End		

Control transfer, Version A:
APDU level message exchange, case 2 command, short APDU

dwFeatures = 0002 0840h
dwProtocols= 0000 0002h

XFR_BLOCK	**(1) request**	→	
wValue: *bLevelParameter* = 00h, bRFU = 00h wIndex: bRFU = 00h, *bInterface* = 00h wLength = 0005h abData = CLA, INS, P1, P2, Le	(1) data	→	
GET_ICC_STATUS	**(2) request**	→	
wValue: wRFU = 0000h wIndex: bRFU = 00h, *bInterface* = 00h wLength = 0001h	(2) data	←	StatusByte: 4xh StatusByte: 10h
if (StatusByte=10h)	Break		
if (StatusByte=4xh)	(**2**) Repeat		
DATA_BLOCK	**(3) request**	→	
wValue: wRFU = 0000h wIndex: bRFU = 00h, *bInterface* = 00h wLength = Le+02h	(3) data	←	Data(Le), SW1, SW2
	End		

Control transfer, Version A:
APDU level message exchange, case 3 command, short APDU

dwFeatures = 0002 0840h
dwProtocols= 0000 0002h

XFR_BLOCK	**(1) request**	→	
wValue: *bLevelParameter* = 00h, bRFU = 00h wIndex: bRFU = 00h, *bInterface* = 00h wLength = Lc + 0005h abData = CLA, INS, P1, P2, Lc, Data(Lc)	(1) data	→	
GET_ICC_STATUS wValue: wRFU = 0000h wIndex: bRFU = 00h, *bInterface* = 00h wLength = 0001h	**(2) request** (2) data	→ ←	 StatusByte: 4xh StatusByte: 20h
if (StatusByte=20h) if (StatusByte=4xh)	Break (**2**) Repeat		
DATA_BLOCK wValue: wRFU = 0000h wIndex: bRFU = 00h, *bInterface* = 00h wLength = 0002h	**(3) request** (3) data	→ ←	 SW1, SW2
	End		

Control transfer, Version A:
APDU level message exchange, case 4 command, short APDU

dwFeatures = 0002 0840h
dwProtocols= 0000 0002h

XFR_BLOCK	**(1) request**	→	
wValue: *bLevelParameter* = 00h, bRFU = 00h wIndex: bRFU = 00h, *bInterface* = 00h wLength = Lc + 0006h abData = CLA, INS, P1, P2, Lc, Data(Lc), Le	(1) data	→	
GET_ICC_STATUS wValue: wRFU = 0000h wIndex: bRFU = 00h, *bInterface* = 00h wLength = 0001h	**(2) request** (2) data	→ ←	 StatusByte: 4xh StatusByte: 10h
if (StatusByte=10h) if (StatusByte=4xh)	Break (**2**) Repeat		
DATA_BLOCK wValue: wRFU = 0000h wIndex: bRFU = 00h, *bInterface* = 00h wLength = Le + 02h	**(3) request** (3) data	→ ←	 Data(Le), SW1, SW2
	End		

Control transfer, Version A:
APDU level message exchange, case 2 command, extended APDU

dwFeatures = 0004 0840h
dwProtocols= 0000 0002h

XFR_BLOCK wValue: *bLevelParameter* = 00h, bRFU = 00h wIndex: bRFU = 00h, *bInterface* = 00h wLength = 0007h abData = CLA, INS, P1, P2, P3=00h, Le	**(1) request →** (1) data →
GET_ICC_STATUS wValue: wRFU = 0000h wIndex: bRFU = 00h, *bInterface* = 00h wLength = 0001h	**(2) request →** (2) data ← StatusByte: 4xh StatusByte: 10h StatusByte: 11h
if (StatusByte=10h OR StatusByte=11h) if (StatusByte=4xh)	Break (**2**) Repeat
if (10h)	Execute the next request
DATA_BLOCK wValue: wRFU = 0000h wIndex: bRFU = 00h, *bInterface* = 00h wLength = BufferLength	**(3) request →** (3) data ← Data(<=BufferLength-2), SW1, SW2 End
if (11h)	Loop the next two requests
DATA_BLOCK wValue: wRFU = 0000h wIndex: bRFU = 00h, *bInterface* = 00h wLength = BufferLength	**(n) request →** (n) data ← Data(BufferLength)
GET_ICC_STATUS wValue: wRFU = 0000h wIndex: bRFU = 00h, *bInterface* = 00h wLength = 0001h	**(n+1) request →** (n+1) data ← StatusByte: 4xh StatusByte: 12h StatusByte: 13h

if (StatusByte=12h)
if (StatusByte=4xh)
if (StatusByte=13h)
	Break
	(**n+1**) Repeat
	(**n**) Repeat

DATA_BLOCK wValue: wRFU = 0000h wIndex: bRFU = 00h, *bInterface* = 00h wLength = BufferLength	**(m) request**	→	
	(m) data	←	Data(<=BufferLength-2), SW1, SW2
	End		

Control transfer, Version A:
APDU level message exchange, case 3 command, extended APDU

dwFeatures = 0004 0840h
dwProtocols= 0000 0002h

XFR_BLOCK wValue: *bLevelParameter* = 00h/01h, bRFU = 00h wIndex: bRFU = 00h, *bInterface* = 00h wLength = BufferLength abData = CLA, INS, P1, P2, P3=00h, Lc, Data(BufferLength-7)	**(1) request →** (1) data →		
GET_ICC_STATUS wValue: wRFU = 0000h wIndex: bRFU = 00h, *bInterface* = 00h wLength = 0001h	**(2) request →** (2) data ←	StatusByte: 4xh StatusByte: 20h StatusByte: 11h	
if (StatusByte=20h OR StatusByte=11h) if (StatusByte=4xh)	Break (**2**) Repeat		
if (StatusByte=10h):	execute the next request		
DATA_BLOCK wValue: wRFU = 0000h wIndex: bRFU = 00h, *bInterface* = 00h wLength = 0002h	**(3) request →** (3) data ←	SW1, SW2	
	End		
if (StatusByte=11h)	Loop the next two requests		
XFR_BLOCK wValue: *bLevelParameter* = 03h/02h, bRFU = 00h wIndex: bRFU = 00h, *bInterface* = 00h wLength = BufferLength abData = Data(BufferLength)	**(n) request →** (n) data →		
GET_ICC_STATUS wValue: wRFU = 0000h wIndex: bRFU = 00h, *bInterface* = 00h wLength = 0001h	**(n+1) request →** data ←	StatusByte: 4xh StatusByte: 20h StatusByte: 13h	

if (StatusByte=20h)
if (StatusByte=13h)
if (StatusByte=4xh)
	Break
	Repeat(**n**)
	(**n+1**) Repeat		

DATA_BLOCK wValue: wRFU = 0000h wIndex: bRFU = 00h, *bInterface* = 00h wLength = 0002h	**(m) request →** (m) data ←	SW1, SW2	
	End		

Version B: Control transfer:
APDU level message exchange, case 1 command, short APDU

dwFeatures = 0002 0840h
dwProtocols= 0000 0002h

XFR_BLOCK wValue: *bLevelParameter* = 00h, *bReserved* = 00h wIndex: bRFU = 00h, *bInterface* = 00h wLength = 0004h abData = CLA, INS, P1, P2	**(1) request** →		
	(1) data →		
DATA_BLOCK wValue: wRFU = 0000h wIndex: bRFU = 00h, *bInterface* = 00h wLength = 0003h	**(2) request** →		
	(2) data ←		80h, *wDelayTime*
	or		
	(2) data ←		00h, SW1, SW2
if (*bResponseType*=00h)	Break		
if (*bResponseType*=80h)	**(2)** Repeat		
	End		

Control transfer, Version B:
APDU level message exchange, case 2 command, short APDU

dwFeatures = 0002 0840h
dwProtocols= 0000 0002h

XFR_BLOCK wValue: *bLevelParameter* = 00h, *bReserved* = 00h wIndex: bRFU = 00h, *bInterface* = 00h wLength = 0005h abData = CLA, INS, P1, P2, Le	**(1) request** →		
	(1) data →		
DATA_BLOCK wValue: wRFU = 0000h wIndex: bRFU = 00h, *bInterface* = 00h wLength = Le + 0003h	**(2) request** →		
	(2) data ←		80h, *wDelayTime*
	or		
	(2) data ←		00h, Data(Le), SW1, SW2
if (*bResponseType*=00h)	Break		
if (*bResponseType*=80h)	**(2)** Repeat		
	End		

Control transfer, Version B:
APDU level message exchange, case 3 command, short APDU

dwFeatures = 0002 0840h
dwProtocols= 0000 0002h

XFR_BLOCK wValue: *bLevelParameter* = 00h, *bReserved* = 00h wIndex: bRFU = 00h, *bInterface* = 00h wLength = Lc + 0005h abData = CLA, INS, P1, P2, Lc, Data(Lc)	**(1) request** → (1) data →		
DATA_BLOCK wValue: wRFU = 0000h wIndex: bRFU = 00h, *bInterface* = 00h wLength = 0003h	**(2) request** → (2) data ←	80h, *wDelayTime*	
	or (2) data ←	00h, SW1, SW2	
if (*bResponseType*=00h)	Break		
if (*bResponseType*=80h)	**(2)** Repeat		
	End		

Control transfer,Version B:
APDU level message exchange, case 4 command, short APDU

dwFeatures = 0002 0840h
dwProtocols= 0000 0002h

XFR_BLOCK wValue: *bLevelParameter* = 00h, *bReserved* = 00h wIndex: bRFU = 00h, *bInterface* = 00h wLength = Lc + 0006h abData = CLA, INS, P1, P2, Lc, Data(Lc), Le	**(1) request** → (1) data →		
DATA_BLOCK wValue: wRFU = 0000h wIndex: bRFU = 00h, *bInterface* = 00h wLength = Le + 0003h	**(2) request** → (2) data ←	80h, *wDelayTime*	
	or (2) data ←	00h, Data(Le), SW1, SW2	
if (*bResponseType*=00h)	Break		
if (*bResponseType*=80h)	**(2)** Repeat		
	End		

Control transfer, Version B:
APDU level message exchange, case 2 command, extended APDU

dwFeatures = 0004 0840h
dwProtocols= 0000 0002h

XFR_BLOCK wValue: *bLevelParameter* = 00h, *bReserved* = 00h wIndex: bRFU = 00h, *bInterface* = 00h wLength = 0007h	**(1) request** →		
abData = CLA, INS, P1, P2, P3=00h, Le	(1) data →		
	Loop the next request		
DATA_BLOCK wValue: wRFU = 0000h wIndex: bRFU = 00h, *bInterface* = 00h	**(2) request** →		
wLength = BufferLength	(2) data	←	80h, *wDelayTime*
	or		
	(2) data	←	00h, Data(Le), SW1, SW2
	or		
	(2) data	←	01h, Data(BufferLength)
	or		
	(2) data	←	03h, Data(BufferLength),
	or		
	(2) data	←	02h, Data(<=BufferLength-2), SW1, SW2
if (*bResponseType*=00h OR *bResponseType*=02)	(**2**) End		
if (*bResponseType*=01h OR *bResponseType*=03)			
XFR_BLOCK wValue: *bLevelParameter* = 10h, *bReserved* = 00h wIndex: bRFU = 00h, *bInterface* = 00h wLength = 0000h	**(3) request** →		
abData = empty	(3) →		
	(2) Repeat		
if (*bResponseType*=80h)	**(2)** Repeat		

Control transfer, Version B:
APDU level message exchange, case 3 command, extended APDU

dwFeatures = 0004 0840h
dwProtocols= 0000 0002h

XFR_BLOCK	**(1)**	**request**	→
wValue: *bLevelParameter* = 00h/01h, *bReserved* = 00h			
wIndex: bRFU = 00h, *bInterface* = 00h			
wLength = BufferLength			
abData = CLA, INS, P1, P2, P3=00h, Lc, Data(BufferLength-7)	(1)	data	→
if (*bLevelParameter*=00h)			
DATA_BLOCK	**(2)**	**request**	→
wValue: wRFU = 0000h			
wIndex: bRFU = 00h, *bInterface* = 00h			
wLength = 0003h			
	(2)	data	← 80h, *wDelayTime*
	(2)	Repeat	
	(2)	data	← 00h, SW1, SW2
	End		
if (*bLevelParameter*=01h)			
DATA_BLOCK	**(3)**	**request**	→
wValue: wRFU = 0000h			
wIndex: bRFU = 00h, *bInterface* = 00h			
wLength = 0003h	(3)	data	← 80h, *wDelayTime*
	(3)	Repeat	
	or		
	(3)	data	← 10h
	Loop the next two requests		
XFR_BLOCK	**(n)**	**request**	→
wValue: *bLevelParameter* = 03h/02h, *bReserved* = 00h			
wIndex: bRFU = 00h, *bInterface* = 00h			
wLength = BufferLength			
abData = Data(BufferLength)	(n)	data	→
DATA_BLOCK	**(n+1)**	**request**	→
wValue: wRFU = 0000h			
wIndex: bRFU = 00h, *bInterface* = 00h			
wLength = 0003h	(n+1)	data	← 80h, *wDelayTime*
	(n+1)	Repeat	
if (*bLevelParameter*=02h)	(n+1)	data	← 00h, SW1, SW2
	(**n+1**)	End	
if (*bLevelParameter*=03h)	(n+1)	data	← 10h
	(n+1)	Repeat	

Annex C
(informative)

Terms and definitions in the USB specification

The terms and definitions listed below are extracted from chapter 2, Terms and Abbreviations of the USB specification:

Transfer
One or more bus transactions to move information between a software client and its function.

TransferType
Determines the characteristics of the data flow between a software client and its function. Four standard types are defined: control, interrupt, bulk and isochronous.

Bulk Transfer
One of the four USB transfer types. Bulk transfers are non-periodic, large bursty communication typically used for a transfer that can use any available bandwidth and can also be delayed until bandwidth is available. See also transfer type.

Control Transfer
One of the four USB transfer types. Control transfers support configuration/command/status type communication between client and function. See also transfer type.

Default pipe
The message pipe created by the USB System Software to pass control control and status information between the host and a USB device's control endpoint zero.

Device Endpoint
A uniquely addressable portion of a USB device that is the source or sink of information in a communication flow between the host and device. See also endpoint address.

Endpoint Address
The combination of an endpoint number and an endpoint direction of a USB device. Each endpoint address supports data transfer in one direction.

Host
The host computer system where the USB Host Controller is installed. This includes the host hardware platform (CPU, bus, etc.) and the operating system in use.

Word
A data element that is two bytes (16bits) in size.

Annex D
(informative)

Class specific descriptor Smart Card device class

The table below describes the class specific descriptor as given in the CCID specification, chapter 3.5

Table 3-5 CCID Class Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	36h	Size of this descriptor, in bytes.
1	*bDescriptorType*	1	??	CCID Functional Descriptor type.
2	*bcdCCID*	2	0100h	CCID Specification Release Number in Binary-Coded decimal (i.e., 2.10 is 0210h).
4	*bMaxSlotIndex*	1		The index of the highest available slot on this device. All slots are consecutive starting at 00h. i.e. 0Fh = 16 slots on this device numbered 00h to 0Fh.
5	*bVoltageSupport*	1		This value indicates what voltages the CCID can supply to its slots. The value is a bitwise OR operation performed on the following values: - 01h 5.0V - 02h 3.0V - 04h 1.8V Other bits are RFU.
6	*dwProtocols*	4	RRRR PPPP	RRRR –Upper Word- is RFU = 0000h PPPP –Lower Word- Encodes the supported protocol types. A '1' in a given bit position indicates support for the associated ISO protocol. 0001h = Protocol T=0 0002h = Protocol T=1 All other bits are reserved and must be set to zero. The field is intended to correspond to the PCSC specification definitions. See PCSC Part3. Table 3-1 Tag 0x0120 Example: 00000003h indicates support for T=0 and T=1.
10	*dwDefaultClock*	4		Default ICC clock frequency in kHz encoded as a little endian integer value. Example: 3.58 MHz is encoded as the integer value 3580. (00000DFCh) This is used in ETU and WWT calculations. It is the clock frequency used when reading the ATR data.

Offset	Field	Size	Value	Description
14	*dwMaximumClock*	4		Maximum supported ICC clock frequency in kHz encoded as little endian integer value. Example: 14.32 MHz is encoded as the integer value 14320. (000037F0h)
18	*bNumClockSupported*	1		The number of clock frequencies that are supported by the CCID. If the CCID does not allow the manual setting of the clock frequency, then this value should be 00h. Otherwise, if the CCID is manual and this value is 00h, the supported clock frequency is assumed to be the default clock frequency defined by dwDefaultClock
19	*dwDataRate*	4		Default ICC I/O data rate in bps encoded as little endian integer Example: 9600 bps is encoded as the integer value 9600. (00002580h)
23	*dwMaxDataRate*	4		Maximum supported ICC I/O data rate in bps Example: 115.2Kbps is encoded as the integer value 115200. (0001C200h)
27	*bNumDataRatesSupported*	1		The number of data rates that are supported by the CCID. If the CCID does not allow the manual setting of the data rate, then this value should be 00h. Otherwise, if the reader is manual and this value is 00h, the supported data rate is assumed to be the default data rate defined by ISO.
28	*dwMaxIFSD*	4		• Indicates the maximum IFSD supported by CCID for protocol T=1.
32	*dwSynchProtocols*	4	RRRR PPPP = 000000 00h	• RRRR-Upper Word- is RFU = 0000h • PPPP-Lower Word- encodes the supported protocol types. A '1' in a given bit position indicates support for the associated protocol. 0001h indicates support for the 2-wire protocol [1)] 0002h indicates support for the 3-wire protocol [1)] 0004h indicates support for the I2C protocol [1)] All other values are outside of this specification, and must be handled by vendor-supplied drivers

1) This release of the specification does not support devices with the 2-wire, 3-wire, and I2C protocol so PPPP = 0000h. This field is intended to be forward compatible with the PCSC specification.

Offset	Field	Size	Value	Description
36	*dwMechanical*	4	0000 0008h	See Note Below. The value is a bitwise OR operation performed on the following values: • 00000000h No special characteristics • 00000001h Card accept mechanism 2) • 00000002h Card ejection mechanism 2) • 00000004h Card capture mechanism 2) • 00000008h Card lock/unlock mechanism • 00000010h Permanently inserted ICC
40	*dwFeatures*	4		This value indicates what intelligent features the CCID has. The value is a bitwise OR operation performed on the following values : • 00000000h No special characteristics • 00000002h Automatic parameter configuration based on ATR data • 00000004h Automatic activation of ICC on inserting • 00000008h Automatic ICC voltage selection • 00000010h Automatic ICC clock frequency change according to parameters • 00000020h Automatic baud rate change according to frequency and FI, DI parameters • 00000040h Automatic parameters negotiation made by the CCID (use of warm resets, cold resets or PPS according to a manufacturer proprietary algorithm to select the communication parameters with the ICC) • 00000080h Automatic PPS made by the CCID according to the current parameters • 00000100h CCID can set ICC in clock stop mode • 00000200h NAD value other than 00 accepted (T=1 protocol in use) • 00000400h Automatic IFSD exchange as first exchange (T=1 protocol in use) Only one of the following values may be present : • 00010000h TPDU level exchanges with CCID • 00020000h Short APDU level exchange with CCID • 00040000h Short and Extended APDU level exchange with CCID Only one of the values 00000040h and 00000080h may be present. When value 00000040h is present the host shall not try to change the FI, DI, and protocol currently selected. When an APDU level for exchanges is selected, one of the values 00000040h or 00000080h must be present, as well as the value 00000002h.

2) These mechanisms of the *dwMechanical* parameter have been included for completeness; however, these functions of motorized CCIDs are not covered by this release of the specification. A future release may attempt to standardize the interface to these mechanical functions.

Offset	Field	Size	Value	Description
44	*dwMaxCCIDMessageLength*	4		For extended APDU level the value shall be between 261 + 10 (header) and 65544 +10, otherwise the minimum value is the *dwMaxPacketSize* of the Bulk-OUT endpoint.
48	*bClassGetResponse*	1		Significant only for CCID that offers an APDU level for exchanges. Indicates the default class value used by the CCID when it sends a Get Response command to perform the transportation of an APDU by T=0 protocol. Value FFh indicates that the CCID echoes the class of the APDU.
49	*bClassEnvelope*	1		Significant only for CCID that offers an extended APDU level for exchanges. Indicates the default class value used by the CCID when it sends an Envelope command to perform the transportation of an extended APDU by T=0 protocol. Value FFh indicates that the CCID echoes the class of the APDU.
50	*wLcdLayout*	2	XXYYh	Number of lines and characters for the LCD display used to send messages for PIN entry. XX: number of lines YY: number of characters per line. XXYY=0000h no LCD.
52	*bPINSupport*	1	00h – 03h	This value indicates what PIN support features the CCID has. The value is a bitwise OR operation performed on the following values : 01h PIN Verification supported 02h PIN Modification supported
53	*bMaxCCIDBusySlots*	1	01h – 0Fh	Maximum number of slots which can be simultaneously busy.

Bibliography

[1] ISO 1177:1985, *Information processing — Character structure for start/stop and synchronous character oriented transmission*

[2] ISO/IEC 10536 (all parts), *Identification cards — Contactless integrated circuit cards — Close coupled cards*

[3] ISO/IEC 14443 (all parts), *Identification cards — Contactless integrated circuit cards — Proximity cards*

[4] ISO/IEC 15693 (all parts), *Identification cards — Contactless integrated circuit cards — Vicinity cards*

ICS 35.240.15

Price based on 50 pages