

Universal Serial Bus

Device Class: Smart Card

ICCD

Specification for USB Integrated Circuit(s) Card Devices

Revision 1.0

April 22nd 2005

Intellectual Property Disclaimer

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

A LICENSE IS HEREBY GRANTED TO REPRODUCE AND DISTRIBUTE THIS SPECIFICATION FOR INTERNAL USE ONLY. NO OTHER LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY OTHER INTELLECTUAL PROPERTY RIGHTS IS GRANTED OR INTENDED HEREBY.

AUTHORS OF THIS SPECIFICATION DISCLAIM ALL LIABILITY, INCLUDING LIABILITY FOR INFRINGEMENT OF PROPRIETARY RIGHTS, RELATING TO IMPLEMENTATION OF INFORMATION IN THIS SPECIFICATION. AUTHORS OF THIS SPECIFICATION ALSO DO NOT WARRANT OR REPRESENT THAT SUCH IMPLEMENTATION(S) WILL NOT INFRINGE SUCH RIGHTS.

Contributors

DONNAT	Francis	Gemplus
DRABCZUK	Nicolas	Axalto
DREWS	Steffen	Philips Semiconductor
FRUHAUF	Serge	STMicroelectronics
LEYDIER	Robert	Axalto
SCHNECKENBURGER	Christian	Infineon
WEISS	Dieter	Giesecke & Devrient

Revision History

Revision	Issue Date	Comment
0.7	May 25 th , 2004	DWG Smart Card, Initial document
0.8g	December 1 st , 2004	Fully reviewed by the SCDWG
0.9	February 3 rd , 2005	Release for public review
1.0	April 22 nd , 2005	Release

Table of Contents

1 Introduction	6
1.1 Related Documents	6
1.2 Terms and Abbreviations	6
1.3 Document Conventions	7
2 Overview	8
3 USB-ICC Functional Characteristics	9
3.1 Communication pipes	9
3.2 Selective suspend	9
3.2.1 Idle detection	9
3.2.2 Idle notification request	9
3.3 Resume	9
4 Standard USB Descriptors	10
4.1 Device	10
4.2 Configuration	10
4.3 Interface	11
5 Smart Card Device Class	12
5.1 Descriptors	12
5.2 USB-ICC Endpoints	14
5.2.1 Bulk-OUT Endpoint	15
5.2.2 Bulk-IN Endpoint	15
5.2.3 Interrupt-IN Endpoint	16
6 Data transfers	17
6.1 Bulk transfers	17
6.1.1 Bulk messages	18
6.1.2 Status and error conditions	24
6.2 Control Transfers	25
6.2.1 Version A	25
6.2.2 Version B	32
6.3 Interrupt transfers	38
6.3.1 Virtual insertion/removal event	38
7 Notation for the state diagrams	39

Figures

Figure 1.3-1 Scope of the USB-ICC specification	8
Figure 6.1-1 State diagram, USB-ICC, bulk transfers, short and extended APDUs	19
Figure 6.2-1 State diagram, USB-ICC, control transfers (Version A), character level	29
Figure 6.2-2 State diagram, USB-ICC, control transfers (Version A), short APDUs	30
Figure 6.2-3 State diagram, USB-ICC, control transfers (Version A), extended APDUs	31
Figure 6.2-4 State diagram, USB-ICC, control transfers (Version B).....	35

Tables

Table 1.3-1 Typographic conventions	7
Table 4.1-1 Specific fields for the USB-ICC in the standard device descriptors	10
Table 4.3-1 Interface Descriptor	11
Table 5.1-1 Smart Card Device Class descriptors returned by an USB-ICC.	12
Table 5.2-1 Configuration of endpoints for a USB-ICC	14
Table 5.2-2 Endpoint descriptor Bulk-OUT	15
Table 5.2-3 Endpoint descriptor Bulk-IN	15
Table 5.2-4 Endpoint descriptor Interrupt-IN.....	16
Table 6.1-1 Bulk-IN and bulk-OUT messages	17
Table 6.1-2 PC_to_RDR_IccPowerOn message	20
Table 6.1-3 RDR_to_PC_DataBlock message containing the ATR.....	20
Table 6.1-4 PC_to_RDR_IccPowerOff message	21
Table 6.1-5 RDR_PC_SlotStatus message	21
Table 6.1-6 PC_to_RDR_XfrBlock message	22
Table 6.1-7 RDR_to_PC_DataBlock message containing a data block	23
Table 6.1-8 Bitmap for bStatus field.....	24
Table 6.1-9 Error codes for <i>bError</i>	24
Table 6.2-1 Class specific requests, Version A.....	25
Table 6.2-2 ICC_POWER_ON.....	26
Table 6.2-3 ICC_POWER_OFF	26
Table 6.2-4 XFR_BLOCK	26
Table 6.2-5 DATA_BLOCK	26
Table 6.2-6 GET_ICC_STATUS	26
Table 6.2-7 Description of the StatusByte.....	28
Table 6.2-8 Class specific requests, Version B.....	32
Table 6.2-9 ICC_POWER_ON.....	33
Table 6.2-10 ICC_POWER_OFF	33
Table 6.2-11 XFR_BLOCK	33
Table 6.2-12 DATA_BLOCK	33
Table 6.2-13 SLOT_STATUS request	34
Table 6.2-14 Data stage of DATA_BLOCK.....	36
Table 6.2-15 Bitmap for bStatus field.....	37
Table 6.2-16 Error codes for <i>bError</i>	37
Table 6.3-1 Interrupt-IN message	38

1 Introduction

This document describes proposed requirements and specifications for Universal Serial Bus (USB) Integrated Circuit(s) Card Devices, USB-ICC.

1.1 Related Documents

Reference	Title	Location
[USB20]	USB 2.0 Specification	http://www.usb.org/developers/docs/
[RECN]	Resistor Engineering Change Notice	http://www.usb.org/developers/docs/
[CCID]	Integrated Circuit(s) Card Interface Devices Specification Revision 1.1	http://www.usb.org/developers/devclass_docs#approved

The following related documents can be ordered through www.iso.org

- ISO/IEC 7816-1; Identification Cards – Integrated circuit(s) cards with contacts Part 1: Physical Characteristics
- ISO/IEC 7816-2; Identification Cards – Integrated circuit(s) cards with contacts Part 2: Dimensions and Locations of the contacts
- ISO/IEC 7816-3; Identification Cards – Integrated circuit(s) cards with contacts Part 3: Electronic signals and transmission protocols
- ISO/IEC 7816-4; Identification Cards – Integrated circuit card Part 4: Organization, security and commands for interchange
- ISO/IEC 7816-12; Identification Cards – Integrated circuit cards Part 12: Cards with contacts: USB electrical interface and operating procedures

1.2 Terms and Abbreviations

The meanings of some words have been stretched to suit the purposes of this document. These definitions are intended to clarify the discussions that follow.

APDU	Application Protocol Data Unit
APDU Command Header	The four byte sequence that begins an APDU; CLA INS P1 P2 (ISO/IEC 7816-4 § 5.3.1)
ATR	Answer To Reset
CCID	Integrated Circuit(s) Cards Interface Devices conforming to the specification for Integrated Circuit(s) Cards Interface Devices
Chip Card	Any of a number of similar devices conforming to ISO/IEC 7816. Used interchangeably with Integrated Circuit(s) Card (ICC) or Smart Card.
Cold RESET	The sequence described in the ISO/IEC 7816-3 §5.3.2. The sequence starts with the ICC powered off.
ICC	Integrated Circuit(s) Card. Used interchangeably with Smart Card.
ICCD	Integrated Circuit(s) Card Devices conforming to this specification. Used interchangeably with USB-ICC.

Interface Device	Terminal communication device or machine to which the ICC is electrically connected during operation [ISO/IEC 7816-3].
ISO/IEC	International Standards Organization/ International Electro technical Commission
Lc	Optional part of the body of a command APDU. Its size is 0, 1 or 3 bytes. The maximum number of bytes present in this body.
Le	Optional part of the body of a command APDU. Its size is 0, 1, 2, or 3 bytes. The maximum number of bytes expected in the data field of the response APDU.
P1, P2	INS parameter of a command header.
P3	INS parameter of a command header. P3 contains Lc or Le
RFU	Reserved for Future Use – Must be set to zero unless stated differently.
Smart Card	Used interchangeably with Chip Card or Integrated Circuit(s) Card (ICC)
T = 0 Command Header	The sequence of five bytes; CLA INS P1 P2 P3 [ISO/IEC 7816-3 § 8.3.2].
TPDU	Transport Protocol Data Unit
USB-ICC	USB Integrated Circuit(s) Card. An ICC providing a USB interface [ISO/IEC 7816-12]. Used interchangeably with ICCD.
Warm RESET	The sequence described in the [ISO/IEC 7816-3 § 5.3.3]. The sequence starts with the ICC already powered.

1.3 Document Conventions

Fields that are larger than a byte are stored in little endian. Little endian is a method of storing data that places the least significant byte of multiple-byte values at lower storage addresses. For example, a 16-bit integer stored in little endian format places the least significant byte at the lower address and the most significant byte at the next address.

This specification uses the following typographic conventions:

Table 1.3-1 Typographic conventions

Example of convention	Description
bValue bcdNam wOther	Placeholder prefixes such as ‘b’, ‘bcd’, and ‘w’ are used to denote placeholder type. For example: ab array of bytes b bits or bytes dependent on context bcd binary-coded decimal bm bit map w word (2 bytes) dw double word (4 bytes)

2 Overview

USB Integrated Circuit(s) Card Devices (USB-ICC) as applied to this document, comprise a selection of similar devices conforming to ISO/IEC 7816 specifications.

This document specifies the USB-related configuration information and communication pipes of an USB-ICC.

Also this document specifies protocols by which a host computer interacts with an USB-ICC.

A USB-ICC is connected to a USB host by means of an USB connection device equipped with a USB plug on one side, and a Vendor Specific interconnect on the opposite side. The USB connection device is out of the scope of this specification.

At any time an USB-ICC can be hot plugged to the bus. At any time an USB-ICC can be disconnected from the bus.

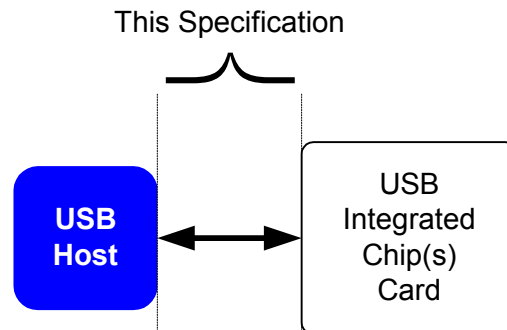


Figure 1.3-1 Scope of the USB-ICC specification

3 USB-ICC Functional Characteristics

3.1 Communication pipes

An 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.

Control transfer mode is supported by two versions, version A and version B. In both cases, the default pipe is used to exchange data between the USB host and the USB-ICC.

Optionally, an USB-ICC may provide an interrupt-IN endpoint to indicate specific events to the host.

3.2 Selective suspend

USB-ICC associated with its driver shall support selective suspend in the following conditions.

3.2.1 Idle detection

The first step in the USB Selective Suspend process is for the Client Driver to determine that the USB-ICC is idle. A USB-ICC is Idle when all commands have had a complete response and when there was no new command during one second.

3.2.2 Idle notification request

The USB Client Driver shall submit an Idle Notification I/O Request Packet (IRP) to the USB HUB driver when it has determined that the USB-ICC is idle.

3.3 Resume

USB-ICC can support remote wake up.

The communication between the application and the USB-ICC is resumed by remote wake up or by the application itself.

4 Standard USB Descriptors

4.1 Device

It is the 12h bytes standard device descriptor as per section 9, “USB Device Framework,” in the *Universal Serial Bus Specification*. For USB-ICC three specific field values are defined, see Table 4.1-1.

Table 4.1-1 Specific fields for the USB-ICC in the standard device descriptors

Offset	Field	Size	Value	Description
4	<i>bDeviceClass</i>	1	00h	Indicates that the device class is specified in the interface descriptor of the device.
5	<i>bDeviceSubClass</i>	1	00h	Reset to zero because <i>bDeviceClass</i> is reset to zero.
6	<i>bDeviceProtocol</i>	1	00h	The device does not use class-specific protocols on the device basis. Instead, it uses class-specific protocols on the interface level.

4.2 Configuration

It is a standard configuration descriptor as per section 9, “USB Device Framework,” in the *Universal Serial Bus Specification*.

It does not contain class-specific information.

4.3 Interface

The values of the interface descriptor fields are listed in Table 4.3-1.

Table 4.3-1 Interface Descriptor

Offset	Field	Size	Value	Description
0	<i>bLength</i>	1	09h	Size of this descriptor in bytes.
1	<i>bDescriptorType</i>	1	04h	INTERFACE descriptor type.
2	<i>bInterfaceNumber</i>	1	Number	Number of this interface. Zero-based value identifying the index in the array of concurrent interfaces supported by this configuration.
3	<i>bAlternateSetting</i>	1	00h	Value used to select this alternate setting for the interface identified in the prior field. Alternate settings are not supported.
4	<i>bNumEndpoints</i>	1	00h 01h 02h 03h	Number of endpoints for a USB-ICC used by this interface (excluding endpoint zero). 00h does not use further endpoint 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	<i>bInterfaceClass</i>	1	0Bh	Smart Card Device Class.
6	<i>bInterfaceSubClass</i>	1	00h	Subclass code.
7	<i>bInterfaceProtocol</i>	1	00h 01h 02h	Protocol code. The given value indicates the transfer mode used for the communication between the host and the USB-ICC. 00h bulk transfers optional interrupt-IN 01h version A, Control transfers, (no interrupt-IN) 02h version B, Control transfers (optional interrupt-IN).
8	<i>iInterface</i>	1	Index	Index of string descriptor this interface.

5 Smart Card Device Class

5.1 Descriptors

The USB-ICC class specific descriptors are mapped on the Smart Card Device Class descriptors published in the Integrated Circuit(s) Cards Interface Devices specification, see § 1.1.

The USB-ICC class specific descriptors are returned by the USB-ICC to set the host's driver in a dedicated mode. The asterisk (*) identifies Fields evaluated by the driver. Other fields are not evaluated by the driver but returned as mentioned for consistency within the Smart Card Class between Integrated Circuit(s) Cards Interface Devices (CCID) and Integrated Circuit(s) Card Devices (USB-ICC or ICCD).

Table 5.1-1 Smart Card Device Class descriptors returned by an USB-ICC.

Offset	Field	Size	Value	Description
0	<i>bLength*</i>	1	36h	Size of this descriptor, in bytes.
1	<i>bDescriptorType*</i>	1	21h	Functional Descriptor type.
2	<i>bcdCCID*</i>	2	0110h	Integrated Circuit(s) Cards Interface Devices Specification Release Number in binary coded decimal. Ex: spec rev 1.1, value 0110h
4	<i>bMaxSlotIndex*</i>	1	00h	Index of the highest available slot. An USB-ICC is regarded as a single slot CCID.
5	<i>bVoltageSupport</i>	1	01h	5.0 Volt, not relevant but fixed to 01h for legacy reason
6	<i>dwProtocols*</i>	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 <i>dwProtocols</i> and <i>dwFeatures</i> are not supported by the USB-ICC. This applies for Bulk Transfer Mode and for Control Transfer mode.
10	<i>dwDefaultClock</i>	4	0000 0DFCh	3.58MHz, not relevant, fixed for legacy reason
14	<i>dwMaximumClock</i>	4	0000 0DFCh	3.58MHz, not relevant, fixed for legacy reason
18	<i>bNumClockSupported</i>	1	00h	Default clock, not relevant, fixed for legacy reason
19	<i>dwDataRate</i>	4	0000 2580h	9600bps, not relevant, fixed for legacy reason
23	<i>dwMaxDataRate</i>	4	0000 2580h	9600bps, not relevant, fixed for legacy reason
27	<i>bNumDataRatesSupported</i>	1	00h	Default data rate, not relevant, fixed for legacy reason

Offset	Field	Size	Value	Description
28	<i>dwMaxIFSD*</i>	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: 00000FEh For T = 0: any value
32	<i>dwSynchProtocols</i>	4	0000 0000h	ISO7816-3, not relevant, fixed to for legacy reason
36	<i>dwMechanical</i>	4	0000 0000h	No special characteristic, not relevant, fixed to for legacy reason
40	<i>dwFeatures*</i>	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 <i>dwProtocols</i>
44	<i>dwMaxCCIDMessageLength*</i>	4		The value shall be between: for bulk transfers: (261 + 10) and (65544 + 10). NOTE: The value 10 is the size of the header for control transfers: 261 and 65544.
48	<i>bClassGetResponse*</i>	1	FFh	Echoes the class of the APDU
49	<i>bClassEnvelope*</i>	1	FFh	Echoes the class of the APDU
50	<i>wLcdLayout</i>	2	0000h	No LCD, not relevant, fixed for legacy reason
52	<i>bPinSupport</i>	1	00h	No PIN pad, not relevant, fixed for legacy reason
53	<i>bMaxCCIDBusySlots*</i>	1	01h	One slot is busy, the USB-ICC is regarded as a single slot CCID.

5.2 USB-ICC Endpoints

An USB-ICC may either communicate with the host using the default control pipe only or it may communicate with the host over message pipes using bulk-IN and bulk-OUT.

Optionally, the USB-ICC may provide an interrupt-IN endpoint used to emulate the behaviour of an Integrated Circuit(s) Cards Interface Device with an ICC inserted or not.

An USB-ICC may have one of the following configurations listed in Table 5.2-1.

Table 5.2-1 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-OUT	no	no	yes
Bulk-IN	no	no	yes
Interrupt-IN	no	optional	optional

5.2.1 Bulk-OUT Endpoint

The Bulk-Out Endpoint is used to send commands and transfer data from the host to the device.

Table 5.2-2 Endpoint descriptor Bulk-OUT

Offset	Field	Size	Value	Description
0	<i>bLength</i>	1	07h	Size of this descriptor in bytes
1	<i>bDescriptorType</i>	1	05h	ENDPOINT descriptor type
2	<i>bEndpointAddress</i>	1	01-0Fh	The address of this endpoint on the USB device. This address is an endpoint number between 1 and 15. Bit 0..3 Endpoint number Bit 4..6 Reserved, reset to 0 Bit 7 0 = Out
3	<i>bmAttributes</i>	1	02h	This is a Bulk endpoint
4	<i>wMaxPacketSize</i>	2	00xyh	Maximum data transfer size can be 8,16,32, or 64 bytes
6	<i>bInterval</i>	1	00h	Does not apply to Bulk endpoints

5.2.2 Bulk-IN Endpoint

The Bulk-In Endpoint is used to send responses and transfer data from the device to the host in reply to commands received on the Command Pipe.

Table 5.2-3 Endpoint descriptor Bulk-IN

Offset	Field	Size	Value	Description
0	<i>bLength</i>	1	07h	Size of this descriptor in bytes
1	<i>bDescriptorType</i>	1	05h	ENDPOINT descriptor type
2	<i>bEndpointAddress</i>	1	81-8Fh	The address of this endpoint on the USB device. This address is an endpoint number between 1 and 15. Bit 0..3 Endpoint number Bit 4..6 Reserved, reset to 0 Bit 7 1 = In
3	<i>bmAttributes</i>	1	02h	This is a Bulk endpoint
4	<i>wMaxPacketSize</i>	2	00xyh	Maximum data transfer size can be 8,16,32, or 64 bytes
6	<i>bInterval</i>	1	00h	Does not apply to Bulk endpoints

5.2.3 Interrupt-IN Endpoint

The Interrupt-IN Endpoint is used optionally to report events.

Table 5.2-4 Endpoint descriptor Interrupt-IN

Offset	Field	Size	Value	Description
0	<i>bLength</i>	1	07h	Size of this descriptor in bytes
1	<i>bDescriptorType</i>	1	05h	Endpoint descriptor type
2	<i>bEndpointAddress</i>	1	81-8Fh	The address of this endpoint on the USB device. This address is an endpoint number between 1 and 15. It must be different from the Bulk-IN endpoint address. Bit 3..0 Endpoint number Bit 6..4 Reserved, reset to 0 Bit 7 1 = In
3	<i>bmAttributes</i>	1	03h	This is an Interrupt endpoint
4	<i>wMaxPacketSize</i>	2	00xyh	Maximum data transfer size (depends on the size of the RDR_to_PC_NotifySlotChange message but it is at least 2h)
6	<i>bInterval</i>	1	xyh	Interval for polling endpoint for data transfers. Expressed in milliseconds. Shall be in the range from 1 to 255. The recommended value is 255.

6 Data transfers

When the USB-ICC is attached to the bus and thereafter has obtained a state where the host may use the functions provided by the USB-ICC, the USB-ICC is designated as "Configured".

The USB-ICC shall be configured before the host sends messages.

The exchange of data between a host and an USB-ICC may be done using control transfers or bulk transfers. For control transfers two implementations are possible version A and version B.

The state of the current execution as depicted in state diagrams Figure 6.1-1, Figure 6.2-1, Figure 6.2-2, Figure 6.2-3 and Figure 6.2-4. shall not be affected by the state of the USB interface engine. For example, bus enumeration and suspend shall not cause any transition.

Also, the state of the current application in the USB-ICC shall not be affected by the state of the USB interface engine. Specifically, when the USB-ICC enters suspend the current state of the application shall not be affected.

The notation for the state diagrams is given in § 7.

6.1 Bulk transfers

This section defines messages for bulk transfers, see Table 6.1-1.

To transmit commands, responses and corresponding data between host and USB-ICC, see Figure 6.1-1, only the following messages shall apply:

Table 6.1-1 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.

6.1.1 Bulk messages

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

- Once the USB-ICC is configured, the host shall submit PC_to_RDR_Icc_Power_Off for the USB-ICC to enter its initial state.

- If the USB-ICC receives a PC_to_RDR_IccPowerOn 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 extension, see Table 6.1-8 the value of *bSeq* 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.

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*.

DWG Smart-Card USB Integrated Circuit(s) Card Devices

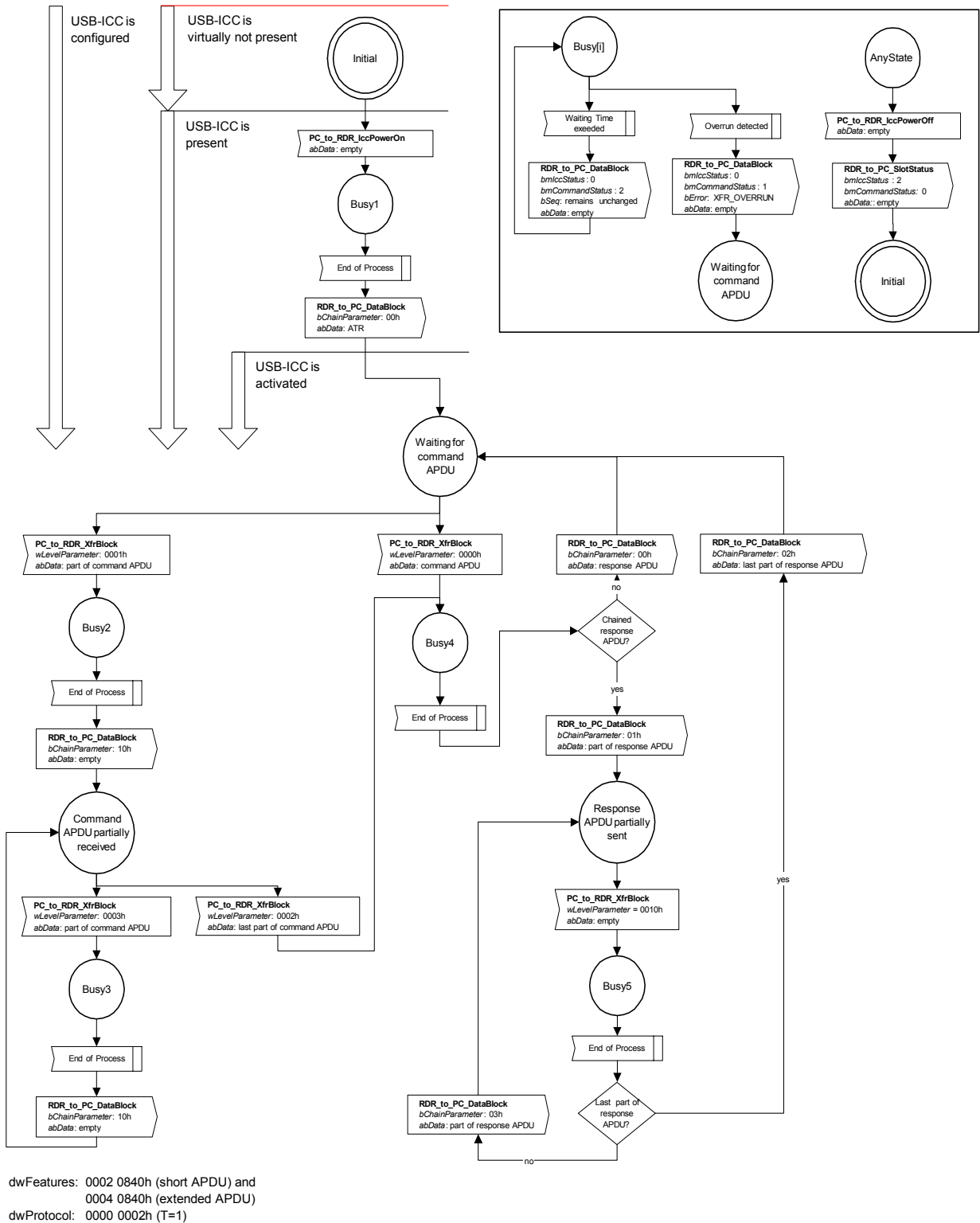


Figure 6.1-1 State diagram, USB-ICC, bulk transfers, short and extended APDUs

6.1.1.1 PC_to_RDR_IccPowerOn and RDR_PC_DataBlock

A PC_to_RDR_IccPowerOn message will return an Answer To Reset (ATR) data in RDR_to_PC_DataBlock.

Each warm reset to a USB- ICC can return a sequentially significant warm ATR. Any subsequent PC_to_RDR_IccPowerOn message will return the next warm ATR.

Table 6.1-2 PC_to_RDR_IccPowerOn message

Offset	Field	Size	Value	Description
0	<i>bMessageType</i>	1	62h	Indicates PC_to_RDR_IccPowerOn
1	<i>dwLength</i>	4	00000000h	Message-specific data length
5	<i>bSlot</i>	1	00h	USB-ICC requires a single slot. Not relevant, fixed for legacy reason
6	<i>bSeq</i>	1	00-FFh	Sequence number for command.
7	<i>bReserved</i>	1	01h	This value shall be 01h
8	<i>abRFU</i>	2	0000h	All other values are reserved for future use

The USB-ICC's response to this command message is the RDR_to_PC_DataBlock response message and the data returned is the Answer To Reset (ATR) data.

Table 6.1-3 RDR_to_PC_DataBlock message containing the ATR

Offset	Field	Size	Value	Description
0	<i>bMessageType</i>	1	80h	Indicates RDR_to_PC_DataBlock
1	<i>dwLength</i>	4		Size of bytes for the ATR
5	<i>bSlot</i>	1	00h	USB-ICC requires a single slot. Not relevant, fixed for legacy reason
6	<i>bSeq</i>	1	Same as Bulk-OUT message	Sequence number for the corresponding command.
7	<i>bStatus</i>	1		USB-ICC Status register as defined in Table 6.1-8
8	<i>bError</i>	1		USB-ICC Error register as defined in Table 6.1-9
9	<i>bChainParameter</i>	1	00h	Indicates that this message contains the complete ATR.
10	<i>abData</i>			ATR

6.1.1.2 PC_to_RDR_IccPowerOff and RDR_PC_SlotStatus

PC_to_RDR_IccPowerOff message sets the USB-ICC to initial conditions.

Table 6.1-4 PC_to_RDR_IccPowerOff message

Offset	Field	Size	Value	Description
0	<i>bMessageType</i>	1	63h	Indicates PC_to_RDR_IccPowerOn
1	<i>dwLength</i>	4	00000000h	Message-specific data length
5	<i>bSlot</i>	1	00h	Slot number for a USB-ICC Not relevant, fixed for legacy reason
6	<i>bSeq</i>	1	00-FFh	Sequence number for command.
7	<i>abRFU</i>	3	000000h	All other values are reserved for future use

The USB-ICC's response to this command message is the RDR_to_PC_SlotStatus response message.

Table 6.1-5 RDR_PC_SlotStatus message

Offset	Field	Size	Value	Description
0	<i>bMessageType</i>	1	81h	Indicates RDR_to_PC_SlotStatus
1	<i>dwLength</i>	4	00000000h	Message-specific data length
5	<i>bSlot</i>	1	00h	USB-ICC requires a single slot. Not relevant, fixed for legacy reason
6	<i>bSeq</i>	1	Same as Bulk-OUT message	Sequence number for the corresponding command.
7	<i>bStatus</i>	1		USB-ICC Status register as defined in Table 6.1-8
8	<i>bError</i>	1		USB-ICC Error register as defined in Table 6.1-9
9	<i>bReserved</i>	1	00h	This value shall be 00h

6.1.1.3 PC_to_RDR_XfrBlock and RDR_PC_DataBlock

This command is used to transmit command APDUs to the USB-ICC.

The *abData* field should never exceed the (*dwMaxCCIDMessageLength* -10) *dwMaxCCIDMessageLength* is defined in the Smart Card Device Class descriptors returned by an USB-ICC, see Table 5.1-1.

Table 6.1-6 PC_to_RDR_XfrBlock message

Offset	Field	Size	Value	Description
0	<i>bMessageType</i>	1	6Fh	Indicates PC_to_RDR_XfrBlock
1	<i>dwLength</i>	4		Size of <i>abData</i> field of this message
5	<i>bSlot</i>	1	00h	USB-ICC requires a single slot. Not relevant, fixed to 00h for legacy reason
6	<i>bSeq</i>	1	00-FFh	Sequence number for command.
7	<i>bReserved</i>	1	00h	Shall be set to 00h
8	<i>wLevelParameter</i>	2		Depends on the exchange level reported by the class descriptor in the <i>dwFeatures</i> field : Character level: Size of expected data to be returned by the bulk-IN endpoint. 0000h Short APDU level 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 continue in the next PC_to_RDR_XfrBlock, 0002h this <i>abData</i> field continues a command APDU and ends the command APDU, 0003h the <i>abData</i> field continues a command APDU and another block is to follow, 0010h empty <i>abData</i> field, continuation of response APDU is expected in the next RDR_to_PC_DataBlock.
10	<i>abData</i>	Byte array		Data block sent to the USB-ICC

The USB-ICC's response to PC_to_RDR_XfrBlock message is RDR_to_PC_DataBlock.

Table 6.1-7 RDR_to_PC_DataBlock message containing a data block

Offset	Field	Size	Value	Description
0	<i>bMessageType</i>	1	80h	Indicates RDR_to_PC_DataBlock
1	<i>dwLength</i>	4		Size of <i>abData</i> field of this message
5	<i>bSlot</i>	1	00h	Slot number for a USB-ICC
6	<i>bSeq</i>	1	Same as Bulk-OUT message	Sequence number for the corresponding command.
7	<i>bStatus</i>	1		USB-ICC Status register as defined in Table 6.1-8
8	<i>bError</i>	1		USB-ICC Error register as defined in Table 6.1-9
9	<i>bChainParameter</i>	1		<p>Depends on the exchange level reported by the class descriptor in <i>dwFeatures</i> field:</p> <p>Character level: 00h Short APDU level: 00h. Extended APDU level:</p> <p>Indicates if the response is complete, to be continued or if the command APDU can continue</p> <p>00h: The response APDU begins and ends in this command</p> <p>01h: The response APDU begins with this command and is to continue</p> <p>02h: This <i>abData</i> field continues the response APDU and ends the response APDU</p> <p>03h: This <i>abData</i> field continues the response APDU and another block is to follow</p> <p>10h: Empty <i>abData</i> field, continuation of the command APDU is expected in next PC_to_RDR_XfrBlock command</p>
10	<i>abData</i>	Byte array		This field contains the data returned by the USB-ICC.

6.1.2 Status and error conditions

The Bulk-IN messages RDR_to_PC_SlotStatus and RDR_to_PC_DataBlock contain status information about the USB-ICC if processed commands 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 (*bmlccStatus*) and the processed command (*bmCommandStatus*). The following two tables give the values for the status and the error codes.

Table 6.1-8 Bitmap for *bStatus* field

Offset	Field	Size	Value	Description
0	<i>bmlccStatus</i>	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)	<i>bmCommandStatus</i>	(2 bits)	0, 1, 2	0 = Processed without error. 1 = Failed, error condition given by <i>bError</i> . 2 = Time extension is requested 3 = RFU
1	<i>bError</i>	1		Error codes

Only the following errors are supported.

Table 6.1-9 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 a hardware error.
	-64 to -127 (C0h to 81h)	User defined
	-3 (FDh), -8 to -14 (F8h to 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 6.1-9
 - offset = if the USB-ICC cannot parse one field in the (10 bytes) 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.

6.2 Control Transfers

This transfer mode can be employed for USB-ICCs that offer low speed functions. This section defines the class specific requests for control transfers. These requests provide the same services to the application layer as bulk transfers.

There are two possible implementations for control transfers, version A and version B.

6.2.1 Version A

6.2.1.1 Specific requests

The following table defines valid values of bRequest.:

Table 6.2-1 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.

6.2.1.2 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 4.3-1.
- 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

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 supporting version A.

Table 6.2-2 ICC_POWER_ON

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100001B	ICC_POWER_ON	wRFU	bRFU <i>bInterface</i>	Length of ATR	ATR

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

Table 6.2-3 ICC_POWER_OFF

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	ICC_POWER_OFF	wRFU	bRFU <i>bInterface</i>	0000h	Empty

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

Table 6.2-4 XFR_BLOCK

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	XFR_BLOCK	bLevelParameter, bRFU	bRFU <i>bInterface</i>	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 6.2-5 DATA_BLOCK

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100001B	DATA_BLOCK	wRFU	bRFU <i>bInterface</i>	Length of data. The value shall be either 2, Le or <i>dwMaxCCIDMessageLength</i> depending of the preceding StatusByte.	Response APDU

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

Table 6.2-6 GET_ICC_STATUS

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100001B	GET_ICC_STATUS	wRFU	bRFU <i>bInterface</i>	0001h	StatusByte

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

6.2.1.3 ATR and data transmission

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 6.2-1, Figure 6.2-2, Figure 6.2-3.

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

- Once the USB-ICC is configured, the host shall submit ICC_POWER_OFF for the USB-ICC to enter its initial state.
- 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 6.2-7, the host should preferably submit ICC_POWER_OFF.
- 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 values in Table 6.2-7.

6.2.1.4 APDU level message exchange, short APDU

In case that the length of the response APDU exceeds the value of wLength in the data 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.

6.2.1.5 Error conditions

Error conditions are returned in the StatusByte, see Table 6.2-7. If the card is not responsive, the value 80h will be returned.

6.2.1.6 Interrupt Transfers

Version A does not use interrupt transfers.

Table 6.2-7 Description of the StatusByte

StatusByte	Description
4xh	<p>busy where x shall be cyclically incremented.</p> <p>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.</p> <p>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.</p>
20h	<p>ready to send status words only</p> <p>Indicates that the data stage of the subsequent DATA_BLOCK will convey SW1-SW2 only.</p>
1yh	<p>if <i>dwProtocols</i> = 00000001h and <i>dwFeatures</i> = 00000840h</p> <p>10h: ready to send data or 10h: ready to receive data</p> <p>The status words are not returned when the value is 10.</p> <p>When GET_ICC_STATUS returns StatusByte = 20h, a subsequent DATA_BLOCK request shall be submitted to obtain the status words.</p> <p>if <i>dwProtocols</i> = 00000002h and <i>dwFeatures</i> = 000z0840h (with z=2 or z=4) the StatusByte has two different functions.</p> <p>When <i>bLevelParameter</i> 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).</p> <p>When <i>bLevelParameter</i> 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):</p> <ul style="list-style-type: none"> 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	<p>mute the card is not responsive</p>
00h	<p>The USB-ICC is ready to receive a command APDU</p>

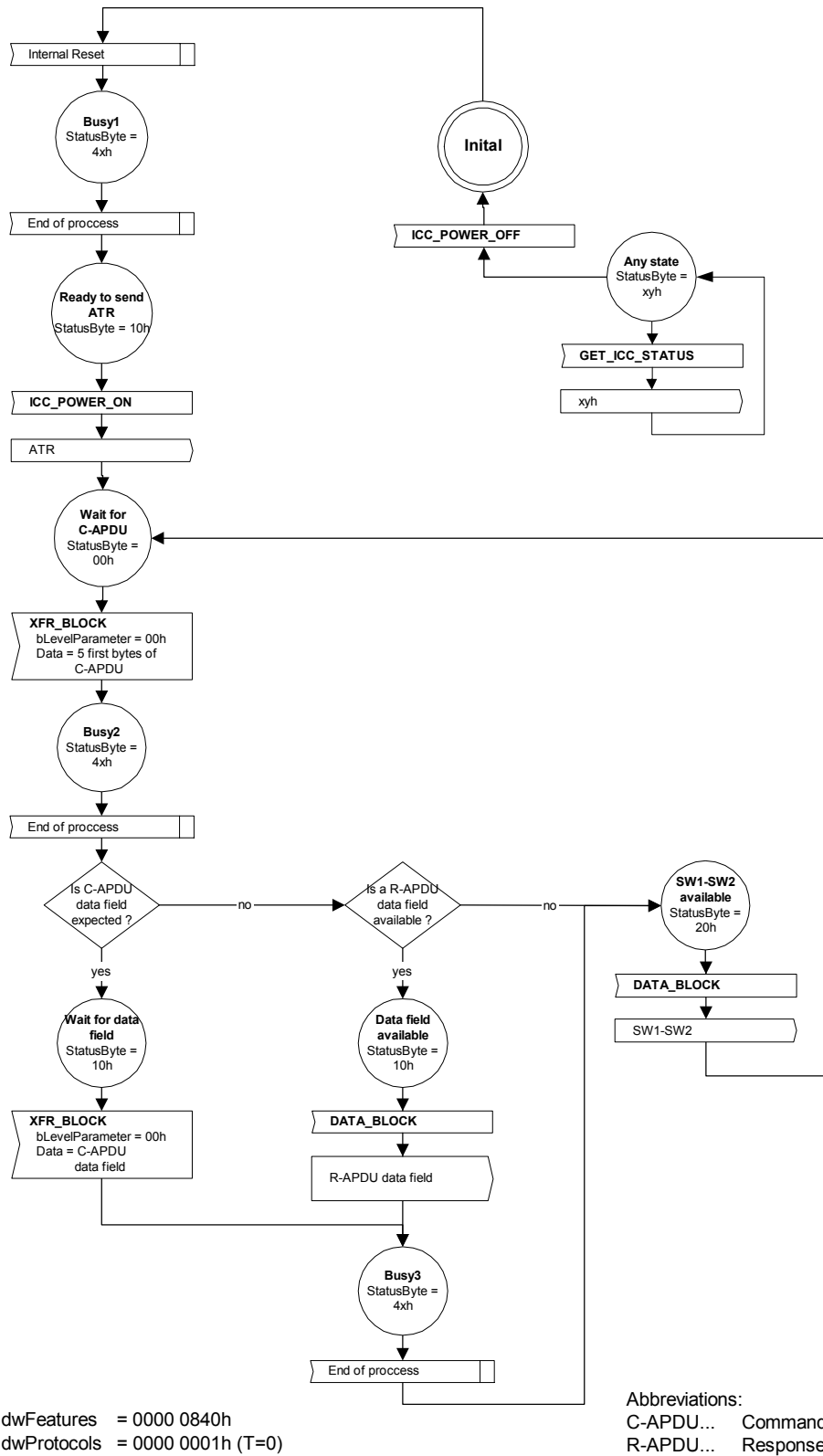
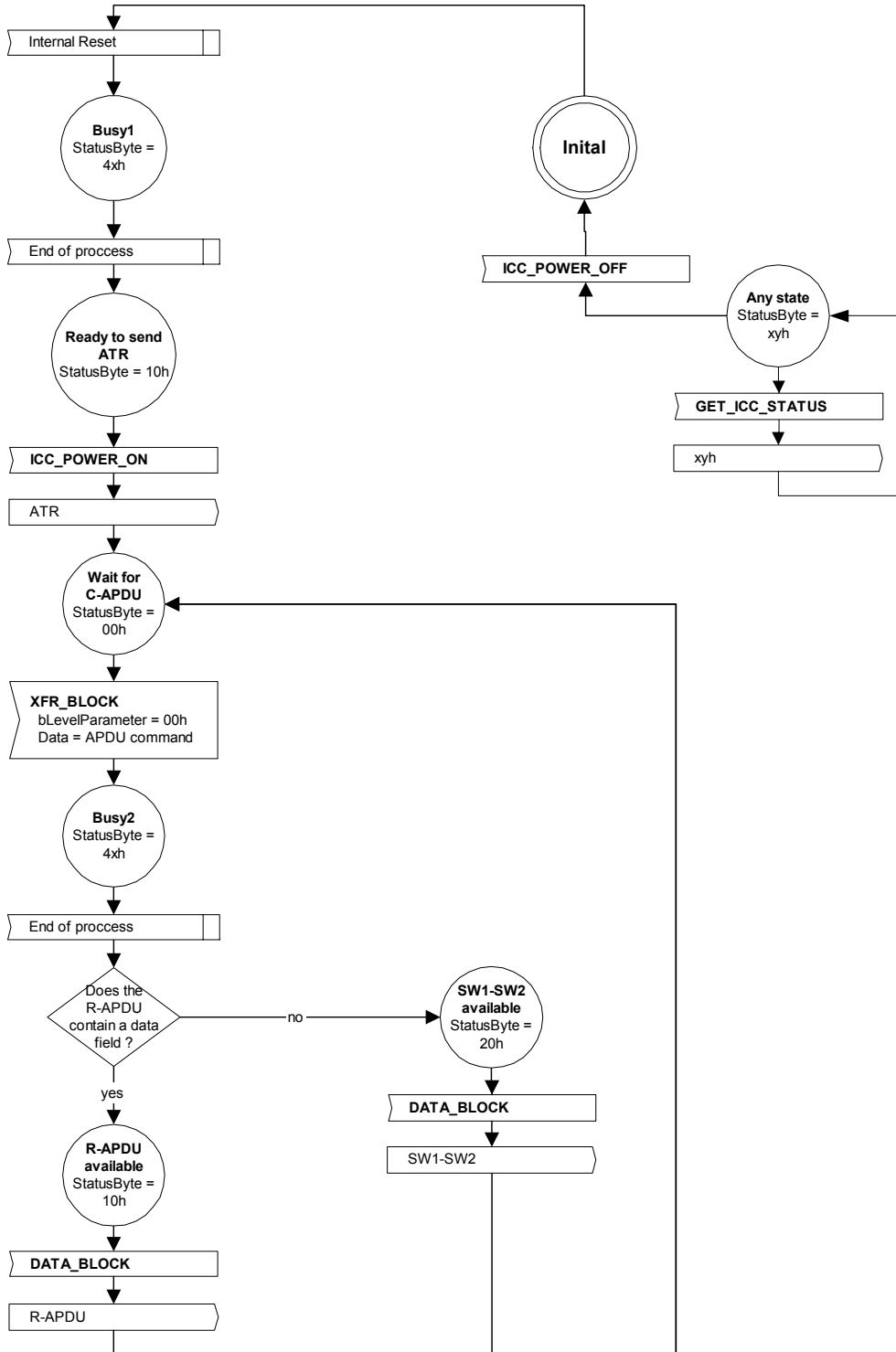


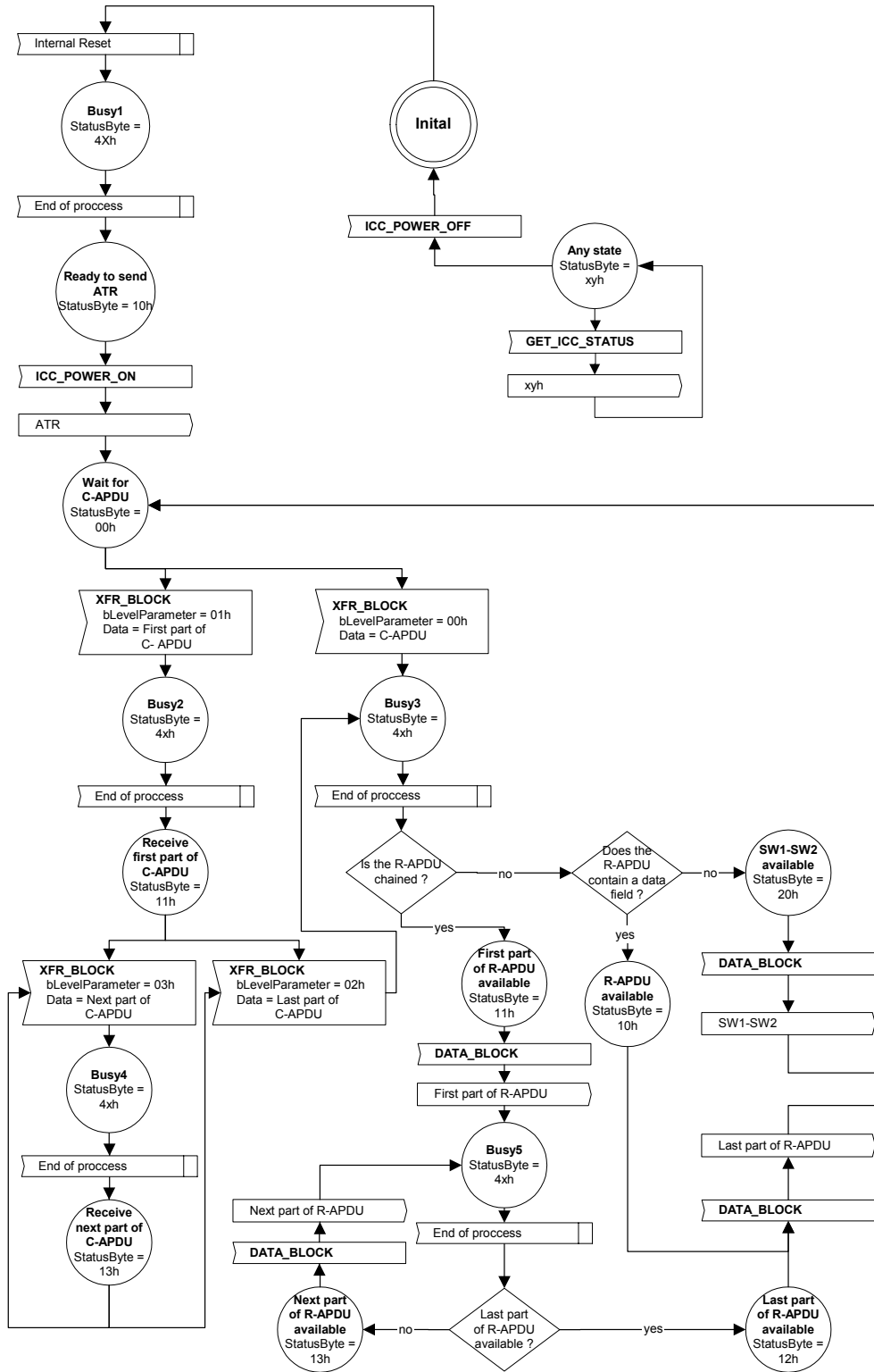
Figure 6.2-1 State diagram, USB-ICC, control transfers (Version A), character level



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

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

Figure 6.2-2 State diagram, USB-ICC, control transfers (Version A), short APDUs



dwFeatures = 0004 0840h
dwProtocols = 0000 0002h

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

Figure 6.2-3 State diagram, USB-ICC, control transfers (Version A), extended APDUs

6.2.2 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.

6.2.2.1 Specific requests

The following table defines valid values of bRequest:

Table 6.2-8 Class specific requests, Version B

bRequest	Value	Direction data stage	Description
ICC_POWER_ON Equivalent to: PC_to_RDR_IccPowerOn	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 Equivalent to: PC_to_RDR_IccPowerOff	63h	OUT	Sets the USB-ICC to initial conditions.
XFR_BLOCK Equivalent to: PC_to_RDR_XfrBlock	65h	OUT	Data transferred from the host to the USB-ICC
DATA_BLOCK Equivalent to: RDR_to_PC_DataBlock	6Fh	IN	Data transferred from the USB-ICC to the host. Also returns information created by the preceding request.
SLOT_STATUS Equivalent to: RDR_to_PC_SlotStatus	81h	IN	The data stage of this command contains <i>bStatus</i> , <i>bError</i> and <i>bReserved</i> . The value for <i>bReserved</i> shall be 00h.

6.2.2.2 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 the Interface Descriptor.
- 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 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 than 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 6.2-9 ICC_POWER_ON

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	ICC_POWER_ON	bRFU bReserved=01h	bRFU <i>bInterface</i>	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 6.2-10 ICC_POWER_OFF

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	ICC_POWER_OFF	wRFU	bRFU <i>bInterface</i>	0000h	Empty

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

Table 6.2-11 XFR_BLOCK

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	XFR_BLOCK	bLevelParameterbReserved=00h	bRFU <i>bInterface</i>	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 bReserved in the low byte.

Table 6.2-12 DATA_BLOCK

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100001B	DATA_BLOCK	wRFU	bRFU <i>bInterface</i>	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.	Response APDU or ATR or status information. Notes: wLength indicates the maximum size of expected data plus one additional byte for bResponseType. If the length of the response APDU is greater than indicated in wLength then the USB-ICC will signal chaining in bResponseType.

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

Table 6.2-13 SLOT_STATUS request

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

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

6.2.2.3 ATR and data transmission

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 diagram

Figure 6.2-4.

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

- Once the USB-ICC is configured, the host shall submit ICC_POWER_OFF for the USB-ICC to enter its initial state.
- 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.

Note When the host has sent ICC_POWER_OFF, the USB-ICC shall enter the state “virtually not present”. The use of interrupt-IN messages for this case is described in § 6.3.

DWG Smart-Card USB Integrated Circuit(s) Card Devices

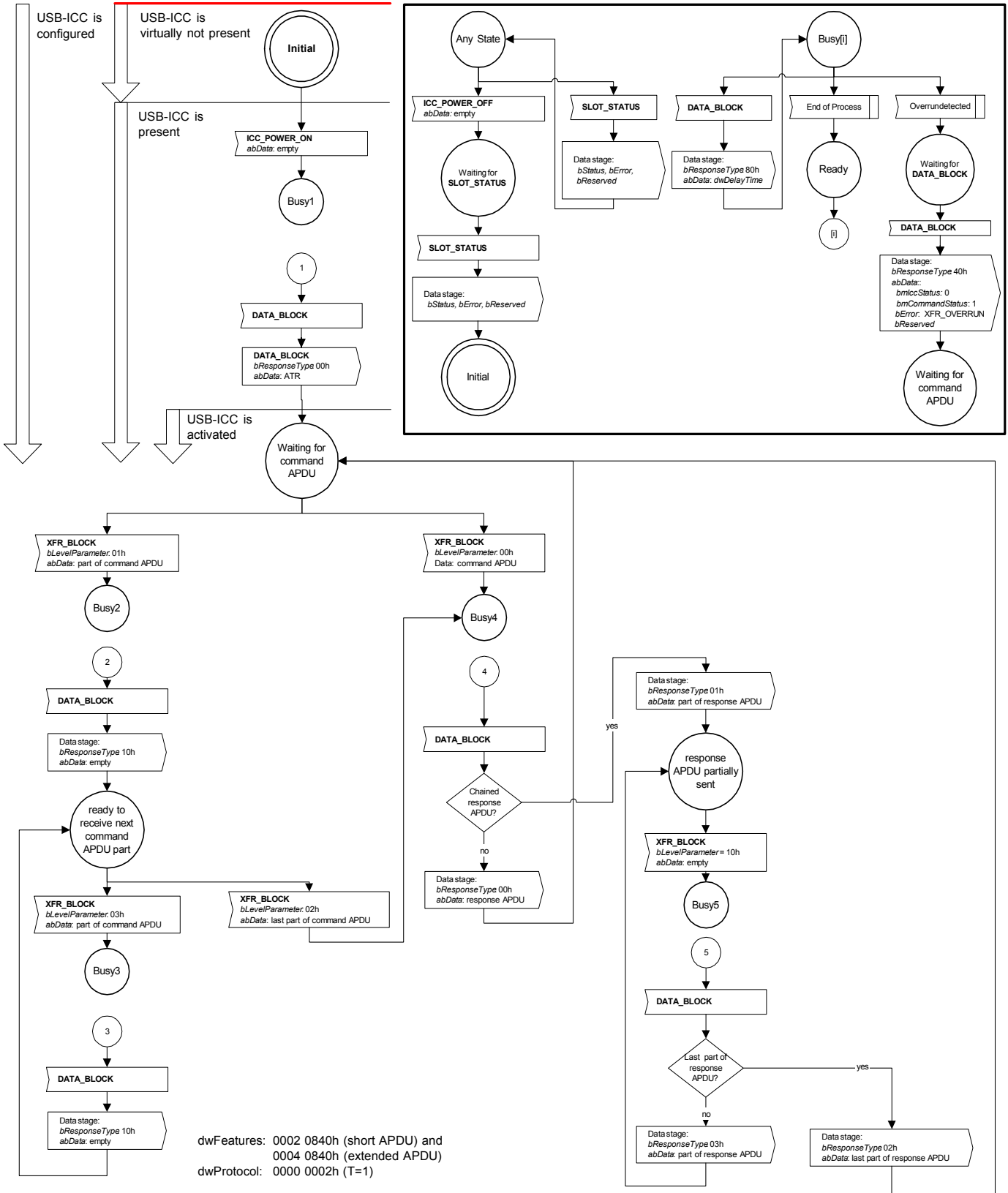


Figure 6.2-4 State diagram, USB-ICC, control transfers (Version B)

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

Table 6.2-14 Data stage of DATA_BLOCK

Offset	Field	Description
0	bResponseType	<p>Indicates the type of information abData field contain:</p> <p>00h: the abData field contains the information created by the preceding request.</p> <p>40h: Status information the abData field contains bStatus, <i>bError</i> and bReserved=00h.</p> <p>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 its best effort.</p> <p>For extended response APDUs:</p> <p>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 7.</p>
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.

6.2.2.4 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_BLOCK request

6.2.2.5 APDU level message exchange, short APDU

In case that the length of the response APDU exceeds the value of wLength in the data 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.

6.2.2.6 Status and error conditions reported by USB requests

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

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 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 6.2-15 Bitmap for *bStatus* field

Offset	Field	Size	Value	Description
0	<i>bmIccStatus</i>	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)	<i>bmCommandStatus</i>	(2 bits)	0, 1	0 = Processed without error. 1 = Failed, error condition given by <i>bError</i> . 2 = RFU 3 = RFU
1	<i>bError</i>	1		Error codes

Only the following errors are supported.

Table 6.2-16 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 a 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.

6.3 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.

Notes:

The driver in combination with the operating system shall provide means to recognize properly card removal.

6.3.1 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 6.3-1 Interrupt-IN message

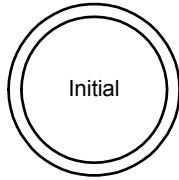
Offset	Field	Size	Value	Description
0	<i>bMessageType</i>	1	50h	Indicates NotifySlotChange.
1	<i>bmSlotIccState</i>	1	000000xyB	<p>The USB-ICC indicates the status (removed or inserted) in the two least significant bits.</p> <p>The least significant bit reports:</p> <p style="padding-left: 20px;">0b= TheUSB-ICC is virtually not present, 1b= The USB-ICC is present.</p> <p>The other bit reports whether the virtual presence of the USB-ICC has changed since the last NotifySlotChange message was sent:</p> <p style="padding-left: 20px;">0b= no change 1b= change</p> <p>All other bits shall be set to zero.</p>

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

- When the USB-ICC exits from the state "Initialized" 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.

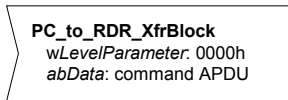
7 Notation for the state diagrams



Initial state in the state diagram



State in the state diagram



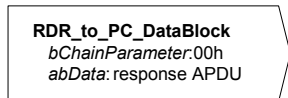
The USB-ICC receives data.

For bulk transfers:

The message type and relevant parameters/values are given.

For control transfers:

The control request and relevant parameters/values are given



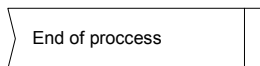
The USB-ICC sends data.

For bulk transfers:

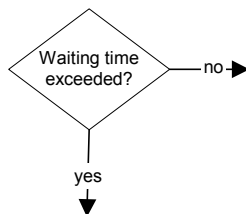
The message type and relevant parameters/values are given. The USB-ICC always initiates the transmission of this message.

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

This page is left intentionally blank.