Embedded USB Host CCID Class Driver User Guide

Version 2.00

For use with USBH CCID Class Driver versions 1.01 and above

Table of Contents

1. System Overview

This chapter contains the fundamental information for this module.

The component sections are as follows:

- Introduction – describes the main elements of the module.
- Feature Check – summarizes the main features of the module as bullet points.
- Packages and Documents – the *Packages* section lists the packages that you need in order to use this module. The *Documents* section lists the relevant user guides.
- Change History – lists the earlier versions of this manual, giving the software version that each manual describes.

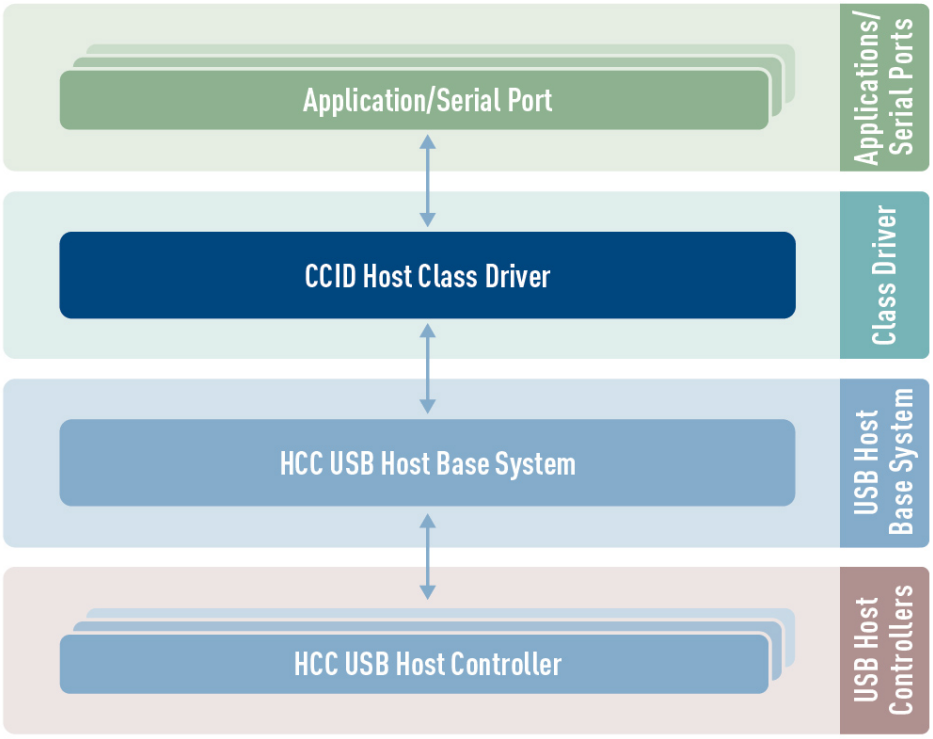
1.1. Introduction

This guide is for those who want to implement an embedded USB host class driver to control Chip Card Interface Device (CCID) USB devices. The CCID protocol is used to connect Integrated Circuit Cards (ICCs) to a computer through an interface device (for example, a card reader) using a standard USB interface.

The module also supports the **pcsc-lite** Application Programming Interface (API) functions for communicating with smart cards and their readers. pcsc-lite implements the Personal Computer/Smart Card (PC/SC) specification.

Examples of ICCs are smart cards and Smart Digital (SD) cards. Examples of CCIDs are card readers, USB smart card reader keyboards that have a slot for accepting a smart card, and USB dongles that contain a Smart Digital (SD) or SIM card. The dongle allows a smart card to be used as a security token for authentication and data encryption.

The system structure is shown in the diagram below:



The lower layer interface is designed to use HCC Embedded's USB Host Interface Layer. This layer is standard over different host controller implementations; this means that the code is unchanged, whichever HCC USB host controller it is interfaced to. For detailed information about this layer, see the *HCC USB Host Base System User Guide* that is shipped with the base system.

When a CCID is connected to a USB host, it may have an ICC "inserted" or not. The CCID identifies to the host its capabilities and requirements, and the host prepares to communicate with it. When the CCID at any time, detects the presence of an ICC, it tells the host. As soon as the host receives information about the inserted ICC, further communications may then take place between the host and the ICC through the CCID.

The CCID exchanges information through a host computer over USB by using CCID messages. These comprise a 10-byte header followed by message-specific data. The standard defines 14 commands that the host computer can use to send data or status and control information in messages. Each command requires at least one response message from the CCID. The term *slot* is used for a physical connection with an CCID.

This package provides two sets of API functions:

- API functions for controlling access to a CCID.
- the pcsc-lite API functions that handle communications with smart card readers.

These are described here, along with separate functions for module management.

Sample code in this guide shows:

- how to initialize the system and perform data transfer.
- how to use the pcsc-lite API functions.

1.2. Feature Check

The main features of the class driver are the following:

- Conforms to the HCC Advanced Embedded Framework.
- Designed for integration with both RTOS and non-RTOS based systems.
- Compatible with all HCC USB host controllers.
- Supports Chip Card Interface Device (CCID) USB devices.
- Supports the pcsc-lite API functions that handle communications with smart card readers. These follow the Personal Computer/Smart Card (PC/SC) specification.
- Supports multiple devices connected simultaneously.

1.3. Packages and Documents

Packages

The table below lists the packages that you need in order to use this module:

Package	Description
hcc_base_doc	This contains the two guides that will help you get started.
usbh_base	The USB host base package. This is the framework used by USB class drivers to communicate over USB using a specific USB host controller package.
usbh_cd_ccid	The USB device CCID host class driver package described by this document.

Documents

For an overview of HCC's embedded USB stacks, see Product Information on the main HCC website.

Readers should note the points in the HCC Documentation Guidelines on the HCC documentation website.

HCC Firmware Quick Start Guide

This document describes how to install packages provided by HCC in the target development environment. Also follow the *Quick Start Guide* when HCC provides package updates.

HCC Source Tree Guide

This document describes the HCC source tree. It gives an overview of the system to make clear the logic behind its organization.

HCC USB Host Base System User Guide

This document defines the USB host base system upon which the complete USB stack is built.

HCC Embedded USB Host CCID Class Driver User Guide

This is this document.

1.4. Change History

This section describes past changes to this manual.

- To view or download manuals, see USB Host PDFs.
- For the history of changes made to the package code itself, see History: usbh_cd_ccid.

The current version of this manual is 2.00. The full list of versions is as follows:

Manual version	Date	Software version	Reason for change
2.00	2020-08-10	1.06	New document format.
1.20	2019-10-21	1.03	Improved API command descriptions.
1.10	2018-10-16	1.03	Corrected **usbh_ccid_pcrdr_setparameters()**. Added two notification codes.
1.00	2018-06-14	1.01	First online release.

2. Source File List

This section describes all the source code files included in the system. These files follow the HCC Embedded standard source tree system, described in the *HCC Source Tree Guide*. All references to file pathnames refer to locations within this standard source tree, not within the package you initially receive.

Note: Do not modify any files except the configuration file.

API Header Files

The following files in the directory **src/api** should be included by any application using the system. These are the only files that should be included by an application using this module. For details of the API functions, see Application Programming Interface.

File	Description
api_pcsc_lite	pcsc-lite API header file.
api_usbh_ccid.h	CCID API header file.

Configuration File

The file **src/config/config_usbh_ccid.h** contains all the configurable parameters of the system. Configure these as required. For details of these options, see Configuration Options.

System Files

These files are in the directory **src/usb-host/class-drivers/ccid**. **These files should only be modified by HCC**.

File	Description
pcsc_lite.c	pcsc-lite API code.
usbh_ccid.c	CCID code.

Version File

The file **src/version/ver_usbh_ccid.h** contains the version number of this module. This version number is checked by all modules that use this module to ensure system consistency over upgrades.

3. Configuration Options

Set the system configuration options in the file **src/config/config_usbh_ccid.h**. This section lists the available options and their default values.

USBH_CCID_MAX_UNITS

The maximum number of units the system can handle. This comprises the maximum number of units per interface and the number of devices connected. The default is 2.

USBH_CCID_MAX_UNITS_PER_INTERFACE

The maximum number of data interfaces per interface. The default is 1.

USBH_CCID_RXBUF_COUNT

The number of buffers to use for receive. The default is 2.

USBH_CCID_RXBUF_SIZE

The size of a receive buffer. The default is 512.

The minimum size for Full Speed (FS) systems is 64 and for High Speed (HS) systems it is 512. The buffer size should always be a multiple of 64 for FS and 512 for HS.

USBH_CCID_BUF_SIZE

The size of a standard buffer. The default is 512.

USBH_CCID_COMBUF_SIZE

The size of the comms. interface buffer. The default is 64.

4. Application Programming Interface

This section documents the Application Programming Interface (API). It includes all the functions that are available to an application program.

4.1. Module Management Functions

The functions are the following:

Function	Description
usbh_ccid_init()	Initializes the module and allocates the required resources.
usbh_ccid_start()	Starts the module.
usbh_ccid_stop()	Stops the module.
usbh_ccid_delete()	Deletes the module and releases the resources it used.

usbh_ccid_init

Use this function to initialize the class driver and allocate the required resources.

> **Note:** You must call this before any other function.

Format

```
int usbh_ccid_init ( void )
```

Arguments

None.

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
Else	See Error Codes.

usbh_ccid_start

Use this function to start the class driver.

> **Note:** You must call **usbh_ccid_init()** before this function.

Format

```
int usbh_ccid_start ( void )
```

Arguments

None.

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
Else	See Error Codes.

usbh_ccid_stop

Use this function to stop the class driver.

Format

```
int usbh_ccid_stop ( void )
```

Arguments

None.

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
Else	See Error Codes.

usbh_ccid_delete

Use this function to delete the class driver and release the associated resources.

Format

```
int usbh_ccid_delete ( void )
```

Arguments

None.

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
Else	See Error Codes.

4.2. Device Management Functions

The functions are the following:

Function	Description
usbh_ccid_pcrdr_iccpoweron()	Activates a slot.
usbh_ccid_pcrdr_iccpoweroff()	Makes a slot inactive.
usbh_ccid_pcrdr_getslotstatus()	Gets the status of a slot.
usbh_ccid_pcrdr_xfrblock()	Transfers a block of data.
usbh_ccid_pcrdr_getparameters()	Gets the parameters for a slot.
usbh_ccid_pcrdr_setparameters()	Sets the parameters for a slot.
usbh_ccid_pcrdr_resetparameters()	Resets the slot parameters to their default values.
usbh_ccid_pcrdr_iccclock()	Stops or restarts the clock.
usbh_ccid_pcrdr_t0apdu()	Changes the parameters used to transport Application Protocol Data Unit (APDU) messages with the T=0 protocol.
usbh_ccid_pcrdr_abort()	Stops any current transfer at the slot and return to a state where the slot is ready to accept a new command pipe Bulk-OUT message.
usbh_ccid_present()	Checks whether a CCID device is connected.
usbh_ccid_register_ntf()	Registers a notification function for a specified event type.

usbh_ccid_pcrdr_iccpoweron

Use this function to activate a slot.

This sends a **PC_to_RDR_IccPowerOn** command to the CCID device.

Format

```
int usbh_ccid_pcrdr_iccpoweron (
    t_ccid_commonparams         cparams,
    t_ccid_powerselect          powerselect,
    t_ccid_rdrpc_datablock *    datablock,
    uint8_t *                   abdata,
    uint32_t                    abdata_len )
```

Arguments

Parameter	Description	Type
cparams	The slot parameters.	t_ccid_commonparams
powerselect	The voltage that is applied to the ICC.	t_ccid_powerselect
datablock	A pointer to the data block.	t_ccid_rdrpc_datablock *
abdata	A pointer to the buffer.	uint8_t *
abdata_len	The size of the *abdata* buffer.	uint32_t

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
USBH_ERROR	Operation failed.

usbh_ccid_pcrdr_iccpoweroff

Use this function to make a slot inactive.

This sends a **PC_to_RDR_IccPowerOff** command to the CCID device.

Format

```
int usbh_ccid_pcrdr_iccpoweroff (
    t_ccid_commonparams          cparams,
    t_ccid_rdrpc_slotstatus *    slotstatus )
```

Arguments

Parameter	Description	Type
cparams	The slot parameters.	t_ccid_commonparams
slotstatus	A pointer to the slot status descriptor.	t_ccid_rdrpc_slotstatus *

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
USBH_ERROR	Operation failed.

usbh_ccid_pcrdr_getslotstatus

Use this function to get the status of a slot.

This sends a **PC_to_RDR_GetSlotStatus** command to the CCID device.

Format

```
int usbh_ccid_pcrdr_getslotstatus (
    t_ccid_commonparams          cparams,
    t_ccid_rdrpc_slotstatus *    slotstatus )
```

Arguments

Parameter	Description	Type
cparams	The slot parameters.	t_ccid_commonparams
slotstatus	On return, a pointer to the slot status descriptor.	t_ccid_rdrpc_slotstatus *

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
USBH_ERROR	Operation failed.

usbh_ccid_pcrdr_xfrblock

Use this function to transfer a block of data.

This sends a **PC_to_RDR_XfrBlock** command to the CCID device.

Format

```
int usbh_ccid_pcrdr_xfrblock (
    t_ccid_commonparams         cparams,
    const uint8_t *             abdata_send,
    uint32_t                    len_abdata_send,
    t_ccid_rdrpc_datablock *    datablock,
    uint8_t *                   abdata_recv,
    uint32_t                    len_abdata_recv )
```

Arguments

Parameter	Description	Type
cparams	The slot parameters.	t_ccid_commonparams
abdata_send	A pointer to the data to send.	uint8_t *
len_abdata_send	The length of the *abdata_send* field.	uint32_t
datablock	A pointer to the data block.	t_ccid_rdrpc_datablock *
abdata_recv	On return, a pointer to the receive buffer.	uint8_t *
len_abdata_recv	On return, the length of the *abdata_recv* buffer.	uint32_t

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
Else	See Error Codes.

usbh_ccid_pcrdr_getparameters

Use this function to get the parameters.

This sends a **PC_to_RDR_GetParameters** command to the CCID device.

Format

```
int usbh_ccid_pcrdr_getparameters (
    t_ccid_commonparams          cparams,
    t_ccid_rdrpc_parameters *    parameters )
```

Arguments

Parameter	Description	Type
cparams	The slot parameters.	t_ccid_commonparams
parameters	On return, a pointer to the parameter structure.	t_ccid_rdrpc_parameters *

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
Else	See Error Codes.

usbh_ccid_pcrdr_setparameters

Use this function to set the parameters.

This sends a **PC_to_RDR_SetParameters** command to the CCID device.

Format

```
int usbh_ccid_pcrdr_setparameters (
    t_ccid_commonparams          cparams,
    uint8_t                      protocolnum,
    t_ccid_rdrpc_parameters *    parameters )
```

Arguments

Parameter	Description	Type
cparams	The slot parameters.	t_ccid_commonparams
protocolnum	The protocol number, 0x00 or 0x01.	uint8_t
parameters	A pointer to the parameter structure.	t_ccid_rdrpc_parameters *

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
Else	See Error Codes.

usbh_ccid_pcrdr_resetparameters

Use this function to reset the slot parameters to their default values.

This sends a **PC_to_RDR_ResetParameters** command to the CCID device.

Format

```
int usbh_ccid_pcrdr_resetparameters (
    t_ccid_commonparams          cparams,
    t_ccid_rdrpc_parameters *    parameters )
```

Arguments

Parameter	Description	Type
cparams	The slot parameters.	t_ccid_commonparams
parameters	On return, a pointer to the parameter structure.	t_ccid_rdrpc_parameters *

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
Else	See Error Codes.

usbh_ccid_pcrdr_iccclock

Use this function to stop or restart the clock.

This sends a **PC_to_RDR_IccClock** command to the CCID device.

Format

```
int usbh_ccid_pcrdr_iccclock (
    t_ccid_commonparams          cparams,
    uint8_t                      bClockCommand,
    t_ccid_rdrpc_slotstatus *    slotstatus )
```

Arguments

Parameter	Description	Type
cparams	The slot parameters.	t_ccid_commonparams
bClockCommand	One of the following: • 0x00 to restart the clock • 0x01 to stop the clock	uint8_t
slotstatus	A pointer to the slot status descriptor.	t_ccid_rdrpc_slotstatus *

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
Else	See Error Codes.

usbh_ccid_pcrdr_t0apdu

Use this function to change the parameters used to transport Application Protocol Data Unit (APDU) messages with the T=0 protocol.

This sends a **PC_to_RDR_T0APDU** command to the CCID device.

Unpowered slots change back to using the default behavior. Any newly inserted ICC has the default behavior until you issue this command for its slot.

This call controls the CLA (class) byte used when issuing a Get Response or Envelope command to the ICC.

Format

```
int usbh_ccid_pcrdr_t0apdu (
    t_ccid_commonparams          cparams,
    uint8_t                      bmChanges,
    uint8_t                      bClassGetResponse,
    uint8_t                      bClassEnvelope,
    t_ccid_rdrpc_slotstatus *    slotstatus )
```

Arguments

Parameter	Description	Type
cparams	The slot parameters.	t_ccid_commonparams
bmChanges	A bitwise OR operation: • Bit 0 is associated with field *bClassGetResponse* • Bit 1 is associated with field *bClassEnvelope*.	uint8_t
bClassGetResponse	A value to force the class byte of the header in a Get Response command. 0xFF indicates that the class byte of the Get Response command echoes the class byte of the APDU.	uint8_t
bClassEnvelope	A value to force the class byte of the header in an Envelope command. 0xFF indicates that the class byte of the Envelope command echoes the class byte of the APDU.	uint8_t
slotstatus	A pointer to the slot status descriptor.	t_ccid_rdrpc_slotstatus *

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
Else	See Error Codes.

usbh_ccid_pcrdr_abort

Use this function to tell the CCID to stop any current transfer at the slot and return to a state where the slot is ready to accept a new command pipe Bulk-OUT message.

This sends a **PC_to_RDR_Abort** command to the CCID device.

Format

```
int usbh_ccid_pcrdr_abort (
    t_ccid_commonparams          cparams,
    t_ccid_rdrpc_slotstatus *    slotstatus )
```

Arguments

Parameter	Description	Type
cparams	The slot parameters.	t_ccid_commonparams
slotstatus	A pointer to the slot status descriptor.	t_ccid_rdrpc_slotstatus *

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
Else	See Error Codes.

usbh_ccid_present

Use this function to check whether a CCID device is connected.

Format

```
int usbh_ccid_present ( t_usbh_unit_id uid )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id

Return Values

Return value	Description
0	No CCID device is present.
1	A CCID device is present.
Else	See Error Codes.

usbh_ccid_register_ntf

Use this function to register a notification function for a specified event type.

When a device is connected or disconnected, or one of the specific events for this type of device occurs, the notification function is called.

Note: It is the user's responsibility to provide any notification functions required by the application. Providing such functions is optional.

Format

```
int usbh_ccid_register_ntf (
    t_usbh_unit_id   uid,
    t_usbh_ntf       ntf,
    t_usbh_ntf_fn    ntf_fn )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
ntf	The notification ID.	t_usbh_ntf
ntf_fn	The notification function to use when an event occurs.	t_usbh_ntf_fn

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
Else	See Error Codes.

4.3. pcsc-lite Functions

The pcsc-lite functions for communicating with smart cards and their readers are the following:

Function	Description
pcsc_init()	Initializes the pcsc-lite API and allocates the required resources.
pcsc_delete()	Deletes the pcsc-lite API and releases the associated resources.
SCardEstablishContext()	Creates an application context.
SCardReleaseContext()	Deletes an application context.
SCardIsValidContext()	Checks whether a context is valid.
SCardConnect()	Establishes a connection to a reader.
SCardDisconnect()	Terminates a connection made by using **SCardConnect()**.
SCardBeginTransaction()	Establishes a temporary exclusive access mode for performing a series of commands in a transaction.
SCardEndTransaction()	Terminates a transaction.
SCardStatus()	Gets the status of a connected reader.
SCardGetStatusChange()	Blocks execution until the current availability of the cards in a specific set of readers changes.
SCardTransmit()	Sends an APDU to the smart card contained in a reader.
SCardListReaders()	Lists the readers currently available on the system.
SCardCancel()	Cancels a specific blocking **SCardGetStatusChange()** function.

The following functions from the standard pcsc-lite API are either not yet supported or do nothing:

Function	Description
SCardReconnect()	Re-establishes a connection to a reader that was previously connected to using **SCardConnect()**. **This function is not yet supported.**
SCardFreeMemory()	Releases memory. **This function does nothing as SCARD_AUTOALLOCATE is not yet supported.**
SCardListReaderGroups()	Lists the reader groups currently available on the system. **This function does nothing as reader groups are not supported.**
SCardControl()	Sends a command directly to the IFD Handler (reader driver) to be processed by the reader. **This function is not yet supported.**
SCardGetAttrib()	Gets an attribute from the IFD Handler (reader driver). **This function is not yet supported.**
SCardSetAttrib()	Sets an attribute of the IFD Handler. **This function is not yet supported.**

pcsc_init

Use this function to initialize the pcsc-lite API and allocate the required resources.

> **Note:** You must call this before any other pcsc-lite function.

Format

```
int pcsc_init ( void )
```

Arguments

None.

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
Else	See SCARD Error Codes.

pcsc_delete

Use this function to delete the pcsc-lite API and release the associated resources.

Format

```
int pcsc_delete ( void )
```

Arguments

None.

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
Else	See SCARD Error Codes.

SCardEstablishContext

Use this function to create an application context.

Note:

- This must be the first function called after **pcsc_init()** in the PC/SC module.
- Difference from the pcsc-lite API: here only one context is returned. Subsequent calls return SCARD_E_NO_MEMORY.

Format

```
LONG SCardEstablishContext (
    DWORD           dwScope,
    LPCVOID         pvReserved1,
    LPCVOID         pvReserved2,
    LPSCARDCONTEXT  phContext )
```

Arguments

Parameter	Description	Type
dwScope	The scope of the establishment. Currently only SCARD_SCOPE_SYSTEM is supported.	DWORD
pvReserved1	Reserved for future use.	LPCVOID
pvReserved2	Reserved for future use.	LPCVOID
phContext	The returned context.	LPSCARDCONTEXT

Return Values

Return value	Description
SCARD_S_SUCCESS	Successful execution.
SCARD_E_INVALID_PARAMETER	*phContext* is null.
SCARD_E_NO_MEMORY	There is no free slot to store the context.
SCARD_E_INVALID_VALUE	*dwScope* is an invalid scope type.

SCardReleaseContext

Use this function to delete an application context.

Format

```
LONG SCardReleaseContext ( SCARDCONTEXT hContext )
```

Arguments

Parameter	Description	Type
hContext	The connection context to delete.	SCARDCONTEXT

Return Values

Return value	Description
SCARD_S_SUCCESS	Successful execution.
SCARD_E_INVALID_HANDLE	*hContext* is an invalid context handle.

SCardIsValidContext

Use this function to check whether a context is valid.

Format

```
LONG SCardIsValidContext ( SCARDCONTEXT hContext )
```

Arguments

Parameter	Description	Type
hContext	The connection context to check.	SCARDCONTEXT

Return Values

Return value	Description
SCARD_S_SUCCESS	The context is valid.
SCARD_E_INVALID_HANDLE	hContext is an invalid context handle.

SCardConnect

Use this function to establish a connection to a reader.

Note: Differences from the pcsc-lite API:

- Here *dwPreferredProtocols* must include SCARD_PROTOCOL_T1 for the call to succeed.
- In *dwShareMode*, SCARD_SHARE_DIRECT is not currently supported.

Format

```
LONG SCardConnect (
    SCARDCONTEXT    hContext,
    LPCSTR          szReader,
    DWORD           dwShareMode,
    DWORD           dwPreferredProtocols,
    LPSCARDHANDLE   phCard,
    LPDWORD         pdwActiveProtocol )
```

Arguments

Parameter	Description	Type
hContext	The connection context returned by **SCardEstablishContext()**.	SCARDCONTEXT
szReader	The name of the reader to connect to.	LPCSTR
dwShareMode	The mode of the connection type: shared or exclusive. (Direct mode is not supported currently.)	DWORD
dwPreferredProtocols	Currently only SCARD_PROTOCOL_T1 is supported.	DWORD
phCard	On return, the connection handle.	LPSCARDHANDLE
pdwActiveProtocol	On return, the established protocol for this connection.	LPDWORD

Return Values

Return value	Description
SCARD_S_SUCCESS	Successful execution.
SCARD_E_INVALID_PARAMETER	*phCard* or *pdwActiveProtocol* is null.
SCARD_E_UNSUPPORTED_FEATURE	The protocol is not supported.
SCARD_E_INVALID_VALUE	Invalid *dwShareMode* sharing mode or *hContext* handle. (SCARD_SHARE_DIRECT is not yet supported.)
SCARD_E_READER_UNAVAILABLE	Cannot communicate with card reader.
SCARD_E_NO_SMARTCARD	No smart card is present.

SCardDisconnect

Use this function to terminate a connection.

Format

```
LONG SCardDisconnect (
    SCARDHANDLE hCard,
    DWORD dwDisposition )
```

Arguments

Parameter	Description	Type
hCard	The connection handle.	SCARDHANDLE
dwDisposition	The dwDisposition reader function to execute: do nothing, reset, eject, or power down.	DWORD

Return Values

Return value	Description
SCARD_S_SUCCESS	Successful execution.
SCARD_E_INVALID_HANDLE	The handle is invalid.
SCARD_E_INVALID_VALUE	*dwDisposition* is invalid.
SCARD_E_NO_SMARTCARD	No smart card is present.

SCardBeginTransaction

Use this function to establish a temporary exclusive access mode for performing a series of commands in a transaction.

> **Note:** Since only one context can be returned by **SCardEstablishContext()**, the user always has exclusive access, even if they do not call **SCardBeginTransaction()**.

Format

```
LONG SCardBeginTransaction ( SCARDHANDLE hCard )
```

Arguments

Parameter	Description	Type
hCard	The connection handle.	SCARDHANDLE

Return Values

Return value	Description
SCARD_S_SUCCESS	Successful execution.
SCARD_E_INVALID_HANDLE	The handle is invalid.

SCardEndTransaction

Use this function to terminate a transaction.

Format

```
LONG SCardEndTransaction (
     SCARDHANDLE    hCard,
     DWORD          dwDisposition )
```

Arguments

Parameter	Description	Type
hCard	The connection handle.	SCARDHANDLE
dwDisposition	The dwDisposition reader function to execute: do nothing, reset, eject, or power down.	DWORD

Return Values

Return value	Description
SCARD_S_SUCCESS	Successful execution.
SCARD_E_INVALID_HANDLE	The handle is invalid.
SCARD_E_INVALID_VALUE	*dwDisposition* is invalid.
SCARD_E_NO_SMARTCARD	No smart card is present.

SCardStatus

Use this function to get the status of a connected reader.

Note:

- If either of the buffer sizes allocated is too small, the function returns with SCARD_E_INSUFFICIENT_BUFFER and the necessary sizes in *pcchReaderLen* and *pcbAtrLen*.
- Difference from the pcsc-lite API: SCARD_AUTOALLOCATE is not currently supported.

Format

```
LONG SCardStatus (
    SCARDHANDLE   hCard,
    LPSTR         szReaderName,
    LPDWORD       pcchReaderLen,
    LPDWORD       pdwState,
    LPDWORD       pdwProtocol,
    LPBYTE        pbAtr,
    LPDWORD       pcbAtrLen )
```

Arguments

Parameter	Description	Type
hCard	The connection handle.	SCARDHANDLE
szReaderName	The friendly name of the reader.	LPSTR
pcchReaderLen	The size of the *szReaderName* buffer.	LPDWORD
pdwState	The current card state.	LPDWORD
pdwProtocol	The current protocol.	LPDWORD
pbAtr	The current ATR (Answer To Reset) of a card in this reader.	LPBYTE
pcbAtrLen	The size of the *pbAtr* buffer.	LPDWORD

Return Values

Return value	Description
SCARD_S_SUCCESS	Successful execution.
SCARD_E_INSUFFICIENT_BUFFER	A buffer is too small; see the note above.
SCARD_E_INVALID_HANDLE	The handle is invalid.
SCARD_E_INVALID_PARAMETER	One or more of the input pointers is NULL.

SCardGetStatusChange

Use this function to block execution until the current availability of the cards in a specific reader changes.

Note:

- If either of the buffer sizes allocated is too small, the function returns with SCARD_E_INSUFFICIENT_BUFFER and the necessary sizes in *pcchReaderLen* and *pcbAtrLen*.
- Difference from the pcsc-lite API: only one *rgReaderStates* structure can be used currently (*cReaders* must be 1).

Format

```
LONG SCardGetStatusChange (
    SCARDCONTEXT          hContext,
    DWORD                 dwTimeout,
    SCARD_READERSTATE *   rgReaderStates,
    DWORD                 cReaders )
```

Arguments

Parameter	Description	Type
hContext	The connection context.	SCARDCONTEXT
dwTimeout	The maximum waiting time for a status change (in milliseconds); use INFINITE to wait forever.	DWORD
rgReaderStates	The structures of readers with current states.	SCARD_READERSTATE *
cReaders	The number of structures. Currently this must be 1.	DWORD

Return Values

Return value	Description
SCARD_S_SUCCESS	Successful execution.
SCARD_E_INVALID_HANDLE	The handle is invalid.
SCARD_E_INVALID_PARAMETER	One or more of the input pointers is NULL.
SCARD_E_UNKNOWN_READER	The reader name given is unknown.
SCARD_E_TIMEOUT	The specified timeout value has expired.
SCARD_E_CANCELLED	The call has been cancelled by a call to **SCardCancel()**.

SCardTransmit

Use this function to send an APDU to the smart card contained in a reader.

Note: The difference from the pcsc-lite API is that here the *pioSendPci* and *pioRecvPci* parameters are not used.

Format

```
LONG SCardTransmit (
    SCARDHANDLE               hCard,
    const SCARD_IO_REQUEST * pioSendPci,
    LPCBYTE                   pbSendBuffer,
    DWORD                     cbSendLength,
    SCARD_IO_REQUEST *        pioRecvPci,
    LPBYTE                    pbRecvBuffer,
    LPDWORD                   pcbRecvLength )
```

Arguments

Parameter	Description	Type
hCard	The card handle.	SCARDHANDLE
pioSendPci	Unused parameter.	SCARD_IO_REQUEST *
pbSendBuffer	The APDU to send to the card.	LPCBYTE
cbSendLength	The length of the APDU.	DWORD
pioRecvPci	Unused parameter.	SCARD_IO_REQUEST *
pbRecvBuffer	The response from the card.	LPBYTE
pcbRecvLength	The length of the response.	LPDWORD

Return Values

Return value	Description
SCARD_S_SUCCESS	Successful execution.
SCARD_E_INVALID_HANDLE	The handle is invalid.
SCARD_E_INVALID_PARAMETER	*pbSendBuffer*, *pbRecvBuffer* or *pcbRecvLength* is null.
SCARD_E_READER_UNAVAILABLE	The reader has been removed.
SCARD_E_NO_SMARTCARD	No smart card is present.
SCARD_E_INSUFFICIENT_BUFFER	*cbRecvLength* was not large enough for the card response. The required size is now in *cbRecvLength*.
SCARD_E_NOT_TRANSACTED	The APDU exchange failed.

SCardListReaders

Use this function to list the readers currently available on the system.

Note: Difference from the pcsc-lite API: here SCARD_AUTOALLOCATE is not currently supported.

Format

```
LONG SCardListReaders (
    SCARDCONTEXT    hContext,
    LPCSTR          mszGroups,
    LPSTR           mszReaders,
    LPDWORD         pcchReaders )
```

Arguments

Parameter	Description	Type
hContext	The connection context.	SCARDCONTEXT
mszGroups	Parameter not used.	LPCSTR
mszReaders	On return, a multi-string with the list of readers.	LPSTR
pcchReaders	On return, the size of the multi-string buffer, including any NULLs. If the application sends *mszGroups* and *mszReaders* as NULL, the size of the buffer needed is returned here.	LPDWORD

Return Values

Return value	Description
SCARD_S_SUCCESS	Successful execution.
SCARD_E_INVALID_HANDLE	*hContext* is an invalid context handle.
SCARD_E_INVALID_PARAMETER	*pcchReaders* is NULL.
SCARD_E_NO_MEMORY	Memory allocation failed.
SCARD_E_INSUFFICIENT_BUFFER	The reader buffer is not large enough. See the note under *pcchReaders* above.
SCARD_E_NO_READERS_AVAILABLE	No readers are available.

SCardCancel

Use this function to cancel a specific blocking **SCardGetStatusChange()** function.

Format

```
LONG SCardCancel ( SCARDCONTEXT hContext )
```

Arguments

Parameter	Description	Type
hContext	The connection context.	SCARDCONTEXT

Return Values

Return value	Description
SCARD_S_SUCCESS	Successful execution.
SCARD_E_INVALID_HANDLE	*hContext* is an invalid context handle.

SCardReconnect

Use this function to re-establish a connection to a reader that was previously connected to using **SCardConnect()**.

Note: Difference from the pcsc-lite API:

- **This function is not supported and returns CARD_E_UNSUPPORTED_FEATURE.**
- In the pcsc-lite API, in a multi application environment an application can reset the card in shared mode. When this occurs any other application trying to access certain commands is returned the value SCARD_W_RESET_CARD. As this HCC implementation only returns one valid context at a time, the above does not apply.

Format

```
LONG SCardReconnect (
    SCARDHANDLE hCard,
    DWORD dwShareMode,
    DWORD dwPreferredProtocols,
    DWORD dwInitialization,
    LPDWORD pdwActiveProtocol )
```

Arguments

Parameter	Description	Type
hCard	The connection handle.	SCARDHANDLE
dwShareMode	The mode of the connection type: shared or exclusive.	DWORD
dwPreferredProtocols	Currently only SCARD_PROTOCOL_T1 is supported.	DWORD
dwInitialization	On return, the connection handle.	DWORD
pdwActiveProtocol	On return, the established protocol for this connection.	LPDWORD

Return Values

Return value	Description
SCARD_E_UNSUPPORTED_FEATURE	This call is not currently functional.

SCardFreeMemory

Use this function to release memory.

> **Note:** Difference from the pcsc-lite API: here **this function does nothing** as SCARD_AUTOALLOCATE is not supported currently.

Format

```
LONG SCardFreeMemory (
    SCARDCONTEXT   hContext,
    LPCVOID        pvMem )
```

Arguments

Parameter	Description	Type
hContext	The connection context to delete.	SCARDCONTEXT
pvMem	A pointer to the allocated memory.	LPCVOID

Return Values

Return value	Description
SCARD_S_SUCCESS	Successful execution.
SCARD_E_INVALID_HANDLE	*hContext* is an invalid context handle.

SCardListReaderGroups

Use this function to list reader groups.

Note: Differences from the pcsc-lite API:

- **This function is not supported and returns CARD_E_UNSUPPORTED_FEATURE.**
- This implementation currently only handles one card reader at a time so reader groups are not used.

Format

```
SCardListReaderGroups (
    SCARDCONTEXT    hContext,
    LPSTR           mszGroups,
    LPDWORD         pcchGroups )
```

Arguments

Parameter	Description	Type
hContext	The context.	SCARDCONTEXT
mszGroups	A list of groups to list readers from.	LPSTR
pcchGroups	The size of the multi-string buffer including NULLs.	LPDWORD

Return Values

Return value	Description
SCARD_E_UNSUPPORTED_FEATURE	This call is not currently functional.

SCardControl

Use this function to send a command directly to the IFD Handler (reader driver) to be processed by the reader.

Note: This function is not yet supported.

SCardGetAttrib

Use this function to get an attribute from the IFD Handler (reader driver).

Note: This function is not yet supported.

SCardSetAttrib

Use this function to set an attribute of the IFD Handler (reader driver).

Note: This function is not yet supported.

4.4. Error Codes

If a function executes successfully it returns with a USBH_SUCCESS code, a value of 0. The following table shows the meaning of the error codes:

Return Code	Value	Description
USBH_SUCCESS	0	Successful execution.
USBH_SHORT_PACKET	1	IN transfer completed with short packet.
USBH_PENDING	2	Transfer still pending.
USBH_ERR_BUSY	3	Another transfer in progress.
USBH_ERR_DIR	4	Transfer direction error.
USBH_ERR_TIMEOUT	5	Transfer timed out.
USBH_ERR_TRANSFER	6	Transfer failed to complete.
USBH_ERR_TRANSFER_FULL	7	Cannot process more transfers.
USBH_ERR_SUSPENDED	8	Host controller is suspended.
USBH_ERR_HC_HALTED	9	Host controller is halted.
USBH_ERR_REMOVED	10	Transfer finished due to device removal.
USBH_ERR_PERIODIC_LIST	11	Periodic list error.
USBH_ERR_RESET_REQUEST	12	Reset request during enumeration.
USBH_ERR_RESOURCE	13	OS resource error.
USBH_ERR_INVALID	14	Invalid identifier/type (HC, EP HDL, and so on).
USBH_ERR_NOT_AVAILABLE	15	Item not available.
USBH_ERR_INVALID_SIZE	16	Invalid size.
USBH_ERR_NOT_ALLOWED	17	Operation not allowed.
USBH_ERROR	18	General error.

4.5. SCARD Error Codes

If a function executes successfully it returns with a SCARD_S_SUCCESS code. The following table shows the meaning of the SCARD error codes:

Return Code	Description
SCARD_S_SUCCESS	Successful execution.
SCARD_F_INTERNAL_ERROR	An internal consistency check failed.
SCARD_E_CANCELLED	The action was cancelled by an **SCardCancel()** request.
SCARD_E_INVALID_HANDLE	The handle supplied is invalid.
SCARD_E_INVALID_PARAMETER	A parameter to the call is invalid.
SCARD_E_INVALID_TARGET	Registry startup information is missing or invalid.
SCARD_E_NO_MEMORY	Not enough memory available to complete this command.
SCARD_F_WAITED_TOO_LONG	An internal consistency timer has expired.
SCARD_E_INSUFFICIENT_BUFFER	The size of buffer allocated by the user is too small.
SCARD_E_UNKNOWN_READER	The reader is not recognized.
SCARD_E_TIMEOUT	The user-specified timeout value has expired.
SCARD_E_SHARING_VIOLATION	The smart card cannot be accessed because of other connections outstanding.
SCARD_E_NO_SMARTCARD	There is no smart card in the reader.
SCARD_E_UNKNOWN_CARD	The specified smart card name is not recognized.
SCARD_E_CANT_DISPOSE	The system could not dispose of the media in the requested manner.
SCARD_E_PROTO_MISMATCH	The requested protocols are incompatible with the protocol currently in use with the smart card.
SCARD_E_NOT_READY	The reader or smart card is not ready to accept commands.
SCARD_E_INVALID_VALUE	One or more of the supplied parameters values could not be properly interpreted.
SCARD_E_SYSTEM_CANCELLED	The action was cancelled by the system, presumably to log off or shut down.
SCARD_F_COMM_ERROR	An internal communications error has been detected.
SCARD_F_UNKNOWN_ERROR	An internal error has been detected, but the source is unknown.
SCARD_E_INVALID_ATR	An ATR obtained from the registry is not a valid ATR string.
SCARD_E_NOT_TRANSACTED	An attempt was made to end a non-existent transaction.
SCARD_E_READER_UNAVAILABLE	The specified reader is not currently available for use.

Return Code	Description
SCARD_P_SHUTDOWN	The operation has been aborted to allow the server application to exit.
SCARD_E_PCI_TOO_SMALL	The PCI Receive buffer was too small.
SCARD_E_READER_UNSUPPORTED	The reader driver does not meet minimal requirements for support.
SCARD_E_DUPLICATE_READER	The reader driver did not produce a unique reader name.
SCARD_E_CARD_UNSUPPORTED	The smart card does not meet minimal requirements for support.
SCARD_E_NO_SERVICE	The Smart card resource manager is not running.
SCARD_E_SERVICE_STOPPED	The Smart card resource manager has shut down.
SCARD_E_UNEXPECTED	An unexpected card error has occurred.
SCARD_E_UNSUPPORTED_FEATURE	This smart card does not support the requested feature.
SCARD_E_ICC_INSTALLATION	No primary provider can be found for the smart card.
SCARD_E_ICC_CREATEORDER	The requested order of object creation is not supported.
SCARD_E_DIR_NOT_FOUND	The identified directory does not exist in the smart card.
SCARD_E_FILE_NOT_FOUND	The identified file does not exist in the smart card.
SCARD_E_NO_DIR	The supplied path does not represent a smart card directory.
SCARD_E_NO_FILE	The supplied path does not represent a smart card file.
SCARD_E_NO_ACCESS	Access to this file is denied.
SCARD_E_WRITE_TOO_MANY	The smart card does not have enough memory to store the information.
SCARD_E_BAD_SEEK	There was an error trying to set the smart card file object pointer.
SCARD_E_INVALID_CHV	The supplied PIN is incorrect.
SCARD_E_UNKNOWN_RES_MNG	An unrecognized error code was returned from a layered component.
SCARD_E_NO_SUCH_CERTIFICATE	The requested certificate does not exist.
SCARD_E_CERTIFICATE_UNAVAILABLE	The requested certificate could not be obtained.
SCARD_E_NO_READERS_AVAILABLE	Cannot find a smart card reader.
SCARD_E_COMM_DATA_LOST	A communications error with the smart card was detected.
SCARD_E_NO_KEY_CONTAINER	The requested key container does not exist on the smart card.
SCARD_E_SERVER_TOO_BUSY	The Smart Card Resource Manager is too busy to complete this operation.

Return Code	Description
SCARD_W_UNSUPPORTED_CARD	The reader cannot communicate with the card, due to ATR string configuration conflicts.
SCARD_W_UNRESPONSIVE_CARD	The smart card is not responding to a reset.
SCARD_W_UNPOWERED_CARD	Power has been removed from the smart card, so further communication is not possible.
SCARD_W_RESET_CARD	The smart card has been reset, so any shared state information is invalid.
SCARD_W_REMOVED_CARD	The smart card has been removed, so further communication is not possible.
SCARD_W_SECURITY_VIOLATION	Access was denied because of a security violation.
SCARD_W_WRONG_CHV	The card cannot be accessed because the wrong PIN was given.
SCARD_W_CHV_BLOCKED	The card cannot be accessed because the maximum number of PIN entry attempts has been reached.
SCARD_W_EOF	The end of the smart card file has been reached.
SCARD_W_CANCELLED_BY_USER	The user pressed "Cancel" on a Smart Card Selection Dialog.
SCARD_W_CARD_NOT_AUTHENTICATED	No PIN was presented to the smart card.

4.6. Types and Definitions

This section describes the *t_usbh_ntf_fn* and the codes that are defined in API Header files.

t_usbh_ntf_fn

The **t_usbh_ntf_fn** definition specifies the format of the notification function. It is defined in the USB host base system in the file **api_usb_host.h**.

Format

```
int  ( * t_usbh_ntf_fn )(
    t_usbh_unit_id   uid,
    t_usbh_ntf       ntf )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
ntf	The notification code.	t_usbh_ntf

Notification Codes

The standard notification codes shown below are defined in the USB host base system in the file **api_usb_host.h**.

Notification	Value	Description
USBH_NTF_CONNECT	1	Connection notification code.
USBH_NTF_DISCONNECT	2	Disconnection notification code.

The additional notification codes provided by this module are as follows:

Notification	Default value	Description
USBH_NTF_CCID_RX	(USBH_NTF_CD_BASE + 0)	Data received.
USBH_NTF_CCID_TX	(USBH_NTF_CD_BASE + 2)	Data successfully transmitted.
USBH_NTF_CCID_SLOT_ACT	(USBH_NTF_CD_BASE + 3)	Slot active.
USBH_NTF_CCID_SLOT_INACT	(USBH_NTF_CD_BASE + 4)	Slot inactive.

t_ccid_commonparams

The *t_ccid_commonparams* structure describes the slot. Its elements are as follows:

Element	Type	Description
uid	uint8_t	The unit ID.
slot	uint8_t	The slot number.
seq	uint8_t	The sequence number of the command. This is used to keep track of commands.

t_ccid_powerselect

The *t_ccid_powerselect* typedef describes the voltage setting the CCID can supply to its slots. Its possible settings are as follows:

Element	Description
AUTOVOLTAGE	Automatic Voltage Selection.
VOLT5_0	5.0 volts.
VOLT3_0	3.0 volts.
VOLT1_8	1.8 volts.

t_ccid_rdrpc_slotstatus

The *t_ccid_rdrpc_slotstatus* structure describes the state of a slot. Its elements are as follows:

Element	Type	Description
bMessageType	uint8_t	The message type.
dwLength	uint32_t	The length of the message, excluding the 10-byte header.
bSlot	uint8_t	The ICC slot being addressed by the message (if the CCID supports multiple slots). The slot number is zero-relative, and is in the range 0 to 0xFF.
bSeq	uint8_t	The sequence number for the command.
bStatus	uint8_t	The slot status.
bError	uint8_t	The slot error.
bClockStatus	uint8_t	One of the following: • 0x00 - clock running. • 0x01 - clock stopped in state L. • 0x02 - clock stopped in state H. • 0x03 - clock stopped in an unknown state.

t_ccid_rdrpc_parameters

The *t_ccid_rdrpc_parameters* structure describes the response message the device sends in response to a parameter function: **usbh_ccid_pcrdr_getparameters()**, **usbh_ccid_pcrdr_setparameters()** or **usbh_ccid_pcrdr_resetparameters()**.

Its elements are as follows:

Element	Type	Description
bMessageType	uint8_t	The message type.
dwLength	uint32_t	The length of the message, excluding the 10-byte header.
bSlot	uint8_t	The ICC slot being addressed by the message (if the CCID supports multiple slots). The slot number is zero-relative, and is in the range 0 to 0xFF.
bSeq	uint8_t	The sequence number for the command.
bStatus	uint8_t	The slot status.
bError	uint8_t	The slot error.
bProtocolNum	uint8_t	Specifies what protocol data structure follows: • 0x00 - structure for protocol T=0. • 0x01 - structure for protocol T=1.
abProtocolDataStructure	t_ccid_abprotocol_datastructure	The Protocol Data Structure.

t_ccid_rdrpc_datablock

The *t_ccid_rdrpc_datablock* structure describes the response message the device sends in response to **usbh_ccid_pcrdr_iccpoweron()**, **usbh_ccid_pcrdr_iccpoweroff()** and **usbh_ccid_pcrdr_xfrblock()**.

Its elements are as follows:

Element	Type	Description
bMessageType	uint8_t	The message type.
dwLength	uint32_t	The length of the message, excluding the 10-byte header.
bSlot	uint8_t	The ICC slot being addressed by the message (if the CCID supports multiple slots). The slot number is zero-relative, and is in the range 0 to 0xFFh.
bSeq	uint8_t	The sequence number.
bStatus	uint8_t	The slot status.
bError	uint8_t	The slot error.
bChainParameter	uint8_t	This depends on the exchange level reported by the class descriptor in its *dwFeatures* field: For Character level, TPDU level, short APDU level, this field is RFU and =0x00. For Extended APDU level - this indicates whether the response is complete, to be continued, or if the command APDU can continue. Values are: • 0x00 - the response APDU begins and ends in this command. • 0x01 - the response APDU begins with this command and is to continue. • 0x02 - this abData field continues the response APDU and ends the response APDU. • 0x03 - this abData field continues the response APDU and another block is to follow. • 0x10 - empty abData field, continuation of the command APDU is expected in next PC_to_RDR_XfrBlock command.

t_ccid_abprotocol_datastructure

The *t_ccid_abprotocol_datastructure* structure describes the protocol.

T=0 Protocol

For T=0 the elements of the structure are as follows:

Element	Type	Description
bmFindexDindex	uint8_t	B7-4 = Index into table 7 in ISO/IEC 7816-3:1997 for selecting a clock rate conversion factor. B3-0 = Index into table 8 in ISO/IEC 7816-3:1997 for selecting a baud rate conversion factor.
bmTCCKST01	uint8_t	The CCID ignores this bit.
bGuardTimeT01	uint8_t	The extra guard time between two characters. Add from 0 to 254 ETUs to the normal guard time of 12 ETU.
bmWaitingIntegersT01	uint8_t	WI for T= 0 used to define WWT.
bClockStop	uint8_t	ICC Clock Stop Support, one of the following: • 00h - stopping the clock is not allowed. • 01h - stop with clock signal Low. • 02h - stop with clock signal High. • 03h - stop with clock either High or Low.
bIFSC	uint8_t	The size of the negotiated IFSC.
bNadValue	uint8_t	0x00 if CCID does not support a value other than the default value.

T=1 Protocol

For T=1 the elements of the structure are as follows:

Element	Type	Description
bmFindexDindex	uint8_t	B7-4 = Index into table 7 in ISO/IEC 7816-3:1997 for selecting a clock rate conversion factor. B3-0 = Index into table 8 in ISO/IEC 7816-3:1997 for selecting a baud rate conversion factor.
bmTCCKST01	uint8_t	The CCID ignores this bit.
bGuardTimeT01	uint8_t	The extra guard time between two characters. Add from 0 to 254 etu to the normal guardtime of 12etu.
bmWaitingIntegersT01	uint8_t	B7-4 = BWI values 0-9 valid B3-0 = CWI values 0-Fh valid
bClockStop	uint8_t	ICC Clock Stop Support, one of the following: • 00h - stopping the clock is not allowed. • 01h - stop with clock signal Low. • 02h - stop with clock signal High. • 03h - stop with clock either High or Low.
bIFSC	uint8_t	The size of the negotiated IFSC.
bNadValue	uint8_t	0x00 if CCID does not support a value other than the default value.

t_ccid_rdrpc_escape

The *t_ccid_rdrpc_escape* structure describes the response to a **usbh_ccid_pcrdr_escape()** call. Its elements are as follows:

Element	Type	Description
bMessageType	uint8_t	The message type.
dwLength	uint32_t	The length of the message, excluding the 10-byte header.
bSlot	uint8_t	The ICC slot being addressed by the message (if the CCID supports multiple slots). The slot number is zero-relative, and is in the range 0 to 0xFF.
bSeq	uint8_t	The sequence number for the command.
bStatus	uint8_t	The slot status.
bError	uint8_t	The slot error.
bRFU	uint8_t	Reserved for Future Use.

t_ccid_rdrpc_datarate_and_clockfrequency

The *t_ccid_rdrpc_datarate_and_clockfrequency* structure describes the response to a **usbh_ccid_pcrdr_setdatarate_and_clockfrequency()** call. Its elements are as follows:

Element	Type	Description
bMessageType	uint8_t	The message type.
dwLength	uint32_t	The length of the message, excluding the 10-byte header.
bSlot	uint8_t	The ICC slot being addressed by the message (if the CCID supports multiple slots). The slot number is zero-relative, and is in the range 0 to 0xFF.
bSeq	uint8_t	The sequence number for the command.
bStatus	uint8_t	The slot status.
bError	uint8_t	The slot error.
bRFU	uint8_t	Reserved for Future Use.
dwClockFrequency	uint32_t	The ICC clock frequency in KHz.
dwDataRate	uint32_t	The ICC data rate in bpd.

SCARD_READERSTATE

The SCARD_READERSTATE, * LPSCARD_READERSTATE structure describes reader states. Its elements are as follows:

Element	Type	Description
szReader	const char *	The reader name.
pvUserData	void *	User-defined data
dwCurrentState	DWORD	The current reader state; see below.
dwEventState	DWORD	The reader state after a state change.
cbAtr	DWORD	The ATR Length, usually MAX_ATR_SIZE.
rgbAtr[MAX_ATR_SIZE]	unsigned char	The ATR Value.

Reader states

The possible *dwCurrentState* values are as follows:

Element	Value	Description
SCARD_STATE_UNAWARE	0x0000	The application is unaware of the current state, and wants to know. Use of this value causes an immediate return from state transition monitoring services.
SCARD_STATE_IGNORE	0x0001	This reader should be ignored.
SCARD_STATE_CHANGED	0x0002	There is a difference between the state the application thinks is set and the state known by the resource manager. When this bit is set, the application may assume a significant state change has occurred on the reader.
SCARD_STATE_UNKNOWN	0x0004	The given reader name is not recognized by the resource manager.
SCARD_STATE_UNAVAILABLE	0x0008	The actual state of this reader is not available. If this bit is set, then all the following bits are clear.
SCARD_STATE_EMPTY	0x0010	There is no card in the reader. If this bit is set, all the following bits will be clear.
SCARD_STATE_PRESENT	0x0020	There is a card in the reader.
SCARD_STATE_EXCLUSIVE	0x0080	The card in the reader is allocated for exclusive use by another application.
SCARD_STATE_INUSE	0x0100	The card in the reader is in use by one or more other applications, but may be connected to in shared mode.
SCARD_STATE_MUTE	0x0200	The card in the reader is unresponsive.

pdwState

The *pdwState* field describes reader states. Its possible values are as follows:

Element	Value	Description
SCARD_ABSENT	0x0002	There is no card in the reader.
SCARD_PRESENT	0x0004	There is a card in the reader, but it is not in position for use.
SCARD_SWALLOWED	0x0008	There is a card in the reader ready for use. The card is not powered on.
SCARD_POWERED	0x0010	Power is being provided to the card, but the reader driver is unaware of its mode.
SCARD_NEGOTIABLE	0x0020	The card has been reset and awaits PTS negotiation.
SCARD_SPECIFIC	0x0040	The card has been reset and specific communication protocols established.

dwDisposition

The possible values of *dwDisposition* are the following:

Value	Action
SCARD_LEAVE_CARD	Do nothing.
SCARD_RESET_CARD	Reset the card (warm reset).
SCARD_UNPOWER_CARD	Power down the card (cold reset).
SCARD_EJECT_CARD	Eject the card.

pdwProtocol

The *pdwProtocol* field describes reader states. Its possible values are as follows:

Element	Value	Description
SCARD_PROTOCOL_T0	0x0001	Use the T=0 protocol.
SCARD_PROTOCOL_T1	0x0002	Use the T=1 protocol.

dwShareMode

The *dwShareMode* field describes share modes. Its possible values are as follows:

Element	Value	Description
SCARD_SHARE_EXCLUSIVE	0x0001	Allows others to share the reader.
SCARD_SHARE_SHARED	0x0002	Does not allow others to share the reader.
SCARD_SHARE_DIRECT	0x0003	**Not currently supported.** Gives direct control of the reader, even without a card.

5. Integration

This section specifies the elements of this package that need porting, depending on the target environment.

5.1. OS Abstraction Layer

All HCC modules use the OS Abstraction Layer (OAL) that allows the module to run seamlessly with a wide variety of RTOSes, or without an RTOS.

The class driver uses the following OAL components:

OAL Resource	Number Required
Tasks	0
Mutexes	4
Events	1

5.2. PSP Porting

The Platform Support Package (PSP) is designed to hold all platform-specific functionality, either because it relies on specific features of a target system, or because this provides the most efficient or flexible solution for the developer.

The class driver makes use of the following standard PSP function:

Function	Package	Element	Description
psp_get_tick_count()	psp_base	psp_tick	Counts the number of ticks.
psp_memcpy()	psp_base	psp_string	Copies a block of memory. The result is a binary copy of the data.
psp_memset()	psp_base	psp_string	Sets the specified area of memory to the defined value.
psp_strncmp()	psp_base	psp_string	Compares two strings of defined length.
psp_strncpy()	psp_base	psp_string	Copies one string to another.

The class driver makes use of the following standard PSP macros:

Macro	Package	Component	Description
PSP_RD_BE32	psp_base	psp_endianness	Reads a 32 bit value stored as big-endian from a memory location.
PSP_WR_BE32	psp_base	psp_endianness	Writes a 32 bit value to be stored as big-endian to a memory location.

6. Sample Code

This section gives example code for the class driver.

6.1. Initialization

This example shows the code used to initialize a USB host with the CCID class driver.

```
/* Initialize USB host with CCID class driver  */

int usb_host_init ( void )
{
    int  rc;
    rc = hcc_mem_init();

    /* Initialize the USB host module */
    if ( rc == USBH_SUCCESS )
    {
        rc = usbh_init();
    }

    /* Initialize the specific USB host controller */
    if ( rc == USBH_SUCCESS )
    {
        rc = usbh_hc_init( 0, usbh_CCID_hc, 0 );
    }

    /* Initialize the CCID Class driver module */
    if ( rc == USBH_SUCCESS )
    {
        rc = usbh_CCID_init();
    }

    /* Start the CCID Class driver */
    if ( rc == USBH_SUCCESS )
    {
        rc = usbh_CCID_start();
    }

    /* Set line coding for the specified CCID serial line */
    if ( rc == USBH_SUCCESS )
    {
        rc = usbh_CCID_set_line_coding( 0, 115200, USBH_CCID_BITS_8,
USBH_CCID_PARITY_NONE, USBH_CCID_STOP_1 );
    }

    /* Start the USB host stack */
    if ( rc == USBH_SUCCESS )
    {
        rc = usbh_start();  /* Start the USB host */
    }

    return rc;

} /* usb_host_init */
```

6.2. Data Transfer

This example shows the code of a simple CCID class driver data handler.

```
/*
** Simple CCID class driver data handler
** usbh_CCID_echo()
*  Send back data received by CCID host.
*/

void usbh_CCID_echo ( void )

{
    uint32_t  len;
    int  rc;

    /* CCID echo demo */

    if ( usbh_CCID_present( 0 ) )    /* Check that device is present */
    {
        rc = usbh_CCID_receive( 0, dbuf, BUF_SIZE, &len );  /* Wait for data */
        if ( ( rc == USBH_SUCCESS ) && ( len != 0 ) )
        {
            usbh_CCID_send( 0, dbuf, len );   /* Echo data back to device */
        }
    }

}
```

6.3. pcsc-lite API Example

This code example uses the pcsc-lite API to:

1. Wait for a card reader to become available.
2. Wait for a card to become available in the reader.
3. Send out an APDU to the card, requesting its UID. The received response is held in *pbRecvBuffer*.

```
LONG rc;
  SCARDCONTEXT context;
  SCARDHANDLE handle;
  DWORD active_proto;

  BYTE pbRecvBuffer[64];
  BYTE pbSendBuffer[64];
  DWORD dwRecvLength;

  DWORD mszReaders_size;
  char mszReaders[32];
  SCARD_READERSTATE rgReaderStates;
  /*****************************************************************************
   * call the necessary init and start functions before running the code below *
   *                                                                           *
   * - hcc_mem_init                                                            *
   * - usbh_init                                                               *
   * - usbh_hc_init                                                            *
   * - usbh_ccid_init                                                          *
   * - pcsc_init                                                               *
   * - usbh_ccid_start                                                         *
   * - usbh_hc_start                                                           *
   * - usbh_start                                                              *
   *****************************************************************************/

  rc = SCardEstablishContext ( SCARD_SCOPE_SYSTEM, NULL, NULL, &context );
  mszReaders_size = 20;
  do
  {
    vTaskDelay( 100u );
    rc = SCardListReaders ( context, NULL, mszReaders, &mszReaders_size );
  } while ( rc == SCARD_E_NO_READERS_AVAILABLE );
  if ( rc == SCARD_S_SUCCESS )
  {
    rgReaderStates.szReader = mszReaders;
    rgReaderStates.dwCurrentState = SCARD_STATE_UNAWARE;
    rc = SCardGetStatusChange ( context, INFINITE, &rgReaderStates, 1 ) ;
    if ( rgReaderStates.dwEventState & SCARD_STATE_EMPTY )
    {
      rgReaderStates.dwCurrentState = rgReaderStates.dwEventState;
      rc = SCardGetStatusChange ( context, INFINITE, &rgReaderStates, 1 );
    }

    if ( rgReaderStates.dwEventState & SCARD_STATE_PRESENT )
    {
```

```
      rc = SCardConnect ( context, mszReaders, SCARD_SHARE_EXCLUSIVE,
SCARD_PROTOCOL_T1, &handle, &active_proto );
      if ( rc == SCARD_S_SUCCESS )
      {
        pbSendBuffer[0] = 0xff;   // getuid APDU
        pbSendBuffer[1] = 0xca;
        pbSendBuffer[2] = 0x00;
        pbSendBuffer[3] = 0x00;
        pbSendBuffer[4] = 0x04;
        dwRecvLength = sizeof ( pbRecvBuffer );
        rc = SCardTransmit ( handle, NULL, pbSendBuffer, 5, NULL, pbRecvBuffer,
&dwRecvLength );
      }

      rc = SCardDisconnect ( handle, SCARD_LEAVE_CARD );
    }
  }
  rc = SCardReleaseContext ( context );
```

7. Version

Version 2.00

For use with USBH CCID Class Driver versions 1.01 and above