Renesas USB MCU

R01AN0643EJ0215
Rev.2.15
Mar 28, 2016

USB Host Communication Device Class Driver (HCDC) using Basic Mini Firmware

Introduction

This document is an application note describing use of the USB Host Communication Device Class Driver (HCDC) build using the USB Basic Mini Firmware of the Renesas USB MCU.

Target Device

RL78/G1C, R8C/3MK, R8C/34K

This program can be used with other microcontrollers that have the same USB module as the above target devices. When using this code in an end product or other application, its operation must be tested and evaluated thoroughly.

This program has been evaluated using the corresponding MCU's Renesas Starter Kit board.

Contents

1. Overview

This application note describes the USB Host Communication Device Class Driver (HCDC) and the sample application using the USB-BASIC-F/W (refer to the Chapter 1.2).

1.1 Functions and Features

The USB Host Communication Device Class Driver (HCDC) conforms to the Abstract Control Model, a Subclass Specification of PSTN Devices, in the USB Communications Device Class specification (CDC from now on). This enables communication with a CDC peripheral device.

This class driver is intended to be used in combination with the USB Basic Mini Firmware provided from Renesas Electronics.

1.2 Related Documents

1. Universal Serial Bus Revision 2.0 specification
2. USB Class Definitions for Communications Devices Revision 1.2
3. USB Communications Class Subclass Specification for PSTN Devices Revision 1.2
   [http://www.usb.org/developers/docs/]
4. User's Manual: Hardware
5. USB Basic Mini Firmware Application Note
   Available from the Renesas Electronics Website

- Renesas Electronics Website
  http://www.renesas.com/
- USB Devices Page
  http://www.renesas.com/prod/usb/

1.3 Terms and Abbreviations

Terms and abbreviations used in this document are listed below.

API	:	Application Program Interface
APL	:	Application program
ACM	:	Abstract Control Model. This is the USB interface subclass used for virtual COM ports, based in the old V.250 (AT) command standard. See PSTN below
CDC	:	Communications devices class
CDCC	:	Communications Devices Class — Communications Class Interface
CDCD	:	Communications Devices Class — Data Class Interface
cstd	:	Prefix for peripheral & host common function of USB-BASIC-F/W
CS+	:	Renesas integration development environment
Data Transfer	:	Generic name of Control transfer, Bulk transfer and Interrupt transfer
HCD	:	Host control driver of USB-BASIC-F/W
HCDC	:	USB Host Communication Device Class Driver (HCDC)
hcdc	:	Prefix for host function & file of HCDC
HDCD	:	Host device class driver (device driver and USB class driver)
HEW	:	High-performance Embedded Workshop
HM	:	Hardware Manual
hstd	:	Prefix for host function of USB-BASIC-F/W
MGR	:	Peripheral device state manager of HCD
PP	:	Pre-processed definition
PSTN	:	Public Switched Telephone Network. Contains the ACM (above) standard. See also Chapter 1.2
RSK	:	Renesas Starter Kit
Scheduler	:	Used to schedule functions, like a simplified OS
Scheduler Macro	:	Used to call a scheduler function
SW1/SW2/SW3	:	User switches on RSK
Task	:	Processing unit
USB	:	Universal Serial Bus
USB-BASIC-FW	:	USB Basic Mini Firmware (Peripheral & Host USB Basic Mini Firmware(USB low level) for Renesas USB MCU)

1.4 How to Read This Document

This document is not intended for reading straight through. Use it first to gain acquaintance with the package, then to look up information on functionality and interfaces as needed for your particular solution.

To get acquainted with the source code, read Chapter 4.3 and note which MCU-specific files you need.

Observe which files belong to the application level "APL". Chapter 5 explains how the sample application works. You will change this to create your own solution.

Understand how execution is divided into tasks, and that these tasks pass messages to one another. This is so that functions (tasks) can execute in the order determined by a scheduler, and not strictly in a predetermined order. This way more important tasks can have a higher priority. Further, tasks are intended to be non-blocking by using a documented callback mechanism. The task mechanism is described in 1.2 above, "USB-BASIC-F/W Application Note". All HCDC tasks are listed in Chapter 4.4 below.

2. How to Register the Class Driver

The class driver, even when modified by the user, must be registered with the USB-BASIC-F/W. Please consult function *usb_hapl_registration()* in *r_usb_hcdc_apl.c* to register the class driver with USB-BASIC-F/W. For details, please refer to the USB-BASIC-F/W application note.

3. Operating Confirmation Environment

3.1 Compiler

The compilers which is used for the operating confirmation are follows.

a. CA78K0R Compiler V.1.71

b. CC-RL Compiler V.1.01

c. IAR C/C++ Compiler for RL78 version 2.10.4

d. KPIT GNURL78-ELF v15.02

e. C/C++ Compiler Package for M16C Series and R8C Family V.6.00 Release 00

3.2 Evaluation Board

The evaluation boards which is used for the operating confirmation are follows.

a. Renesas Starter Kit for RL78/G1C (Product No: R0K5010JGC001BR)

b. R8C/34K Group USB Host Evaluation Board (Product No: R0K5R8C34DK2HBR)

4. Software Configuration

4.1 Module Configuration

Figure 4.1 shows the structure of the HCDC software modules. Table 4-1 lists the modules and an overview of each.

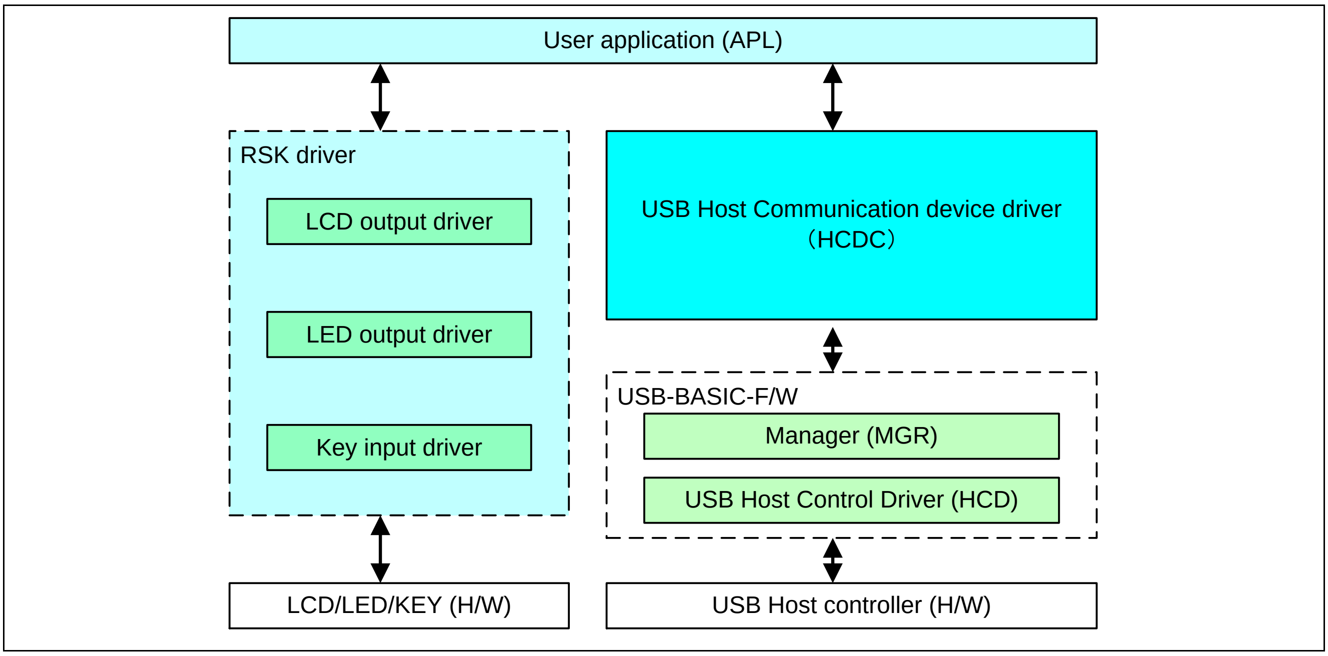

Figure 4.1 Module Structure

Table 4-1 Module Function Descriptions

Module Name	Description	Notes
APL	The user application program. The switches change the UART setting of the CDC device. The LCD displays the CDC device state.	Created by the customer.
HCDC	The registered device class driver checks operation of a connected device. The USB-BASIC-F/W notifies APL whether the connected device corresponds to the CDC class. The following data transfers are requested by APL of USB-BASIC-F/W. 1) Control of connected device by CDC requests 2) State confirmation of connected device by CDC notifications 3) Data transfer with connected device The transfer result is notified to APL via a pre-registered callback function.	
USB-BASIC-F/W	The USB Basic Mini Firmware (Host Hardware Control & Device state Management)	

4.2 Overview of Application Program Functions

The main functions of the host demo application are to.

1. Set the serial state as baud rate etc for the connected USB peripheral.
2. The sample APL loops back (returns) the received data back to the USB peripheral. In other words the HCDC APL echos received user data back to the USB pripheral.
3. Make changes to the baud rate when the user presses SW2 and SW3 on the USB evaluation board. Please do not change transmission speed during a data transmission.

Switch input operation is described in Table 4-2.

Table 4-2 User switch input operation

Switch Function	Description	Switch Number
Baud Rate Selection	To select communication speed (1200→2400→4800→….)	SW2
Baud Rate Setting	Activate the selected baud rate.	SW3

4.3 File Configuration List

4.3.1 Folder Structure

The folder structure of the files supplied with the device class is shown below.

The source code used depends on the MCU and its evaluation board, and is stored in the repecctive hardware resource folder (***devicename****src**HwResource*).

```
workspace
    ＋ [ RL78/G1C / R8C ]
        ＋[ CCRL / CS+ / IAR / e2 studio / HEW ]
          ＋[ RL78G1C / R8C3MK / R8C34K ]
               ＋ HOST                                          Build result
               ＋ src
                 ＋――― CDCFW [ Communication Device  Class driver ]          See Table 4-3
                 |          ＋――― inc                          Common header file of CDC driver
                 |          ＋――― src                          CDC driver
                 ＋―――SmplMain [ Sample Application ]
                 |          ＋――― APL                          Loop back application
                 ＋―――USBSTDFW [Common USB code that is used by all USB firmware ]
                 |          ＋――― inc                          Common header file of USB driver
                 |          ＋――― src                          USB driver
                 ＋――― HwResource [Hardware access layer; to initialize the MCU ]
                              ＋――― inc                        Common header file of hardware resource
                              ＋―――src                         Hardware resource
```

[Note]

a. The project for CA78K0R compiler is stored under the CS+ folder.

b. The project for KPIT GNU compiler is stored under the e^2 studio folder.

c. Refer to **10 Using the e2 studio project with CS+** section when using CC-RL compiler on CS+.

4.3.2 File Structure

Table 4-3 shows the HCDC file structure .

Table 4-3 File Structure

Folder	File Name	Description	Notes
CDCFW/inc	r_usb_class_usrcfg.h	USB host CDC user definitions	
CDCFW/inc	r_usb_hcdc_define.h	HCDC type definitions and macro definitions	
CDCFW/inc	r_usb_hcdc_api.h	HCDC API function prototypes	
CDCFW/src	r_usb_hcdc_api.c	HCDC API functions	
CDCFW/src	r_usb_hcdc_driver.c	HCDC driver functions	
SmplMain	main.c	Main loop processing	
SmplMain/APL	r_usb_hcdc_apl.c	Sample application program	

4.4 System Resources

4.4.1 System Resource Definitions

Table 4-4 lists the Task IDs and the task priorities used when registering the tasks with the scheduler.

These are defined in the *r_usb_ckernelid.h* header file.

Table 4-4 List of Scheduler Registration IDs

Scheduler registration task	Description	Notes
USB_HCDC_TSK	**HCDC** (R_usb_hcdc_task) Task ID: USB_HCDC_TSK Task priority: 2	
USB_HCDCSMP_TSK	**APL** (usb_hcdc_main_task) Task ID: USB_HCDCSMP_TSK Task priority: 3	
USB_HCD_TSK	**HCD** (R_usb_hstd_HcdTask) Task ID: USB_HCD_TSK Task priority: 0	
USB_MGR_TSK	**MGR** (R_usb_hstd_MgrTask) Task ID: USB_MGR_TSK Task priority: 1	
Mailbox ID / Default receive task	**Message description**	**Notes**
USB_HCDC_MBX / USB_HCDC_TSK	HCDC -> HCDC / APL -> HCDC mailbox ID	
USB_HCDCSMP_MBX / USB_HCDCSMP_TSK	HCDC -> APL mailbox ID	
USB_HCD_MBX / USB_HCD_TSK	HCD task mailbox ID	
USB_MGR_MBX / USB_MGR_TSK	MGR task mailbox ID	

5. Host CDC Sample Application Program (APL)

The host demo application performs loopback communication of USB user data when connected to a CDC device. The HCDC application complies with the Abstract Control Model subclass as specified in the USB Communications Device Class specification and its PSTN subclass specification. See Chapter 1.2, items 2 and 3.

5.1 Operating Environment

Figure 5.1 below shows a sample operating environment for the software. If a PC, the rightmost box in the figure, does not have a serial port, two chained USB-serial converters can be used instead of one. Note that many USB-serial converters are of type Vendor class, and not strict CDC class devices (CDC ACM). *To be able to use such converters the HCDC code needs to change according to Chapter 5.4 to work.*

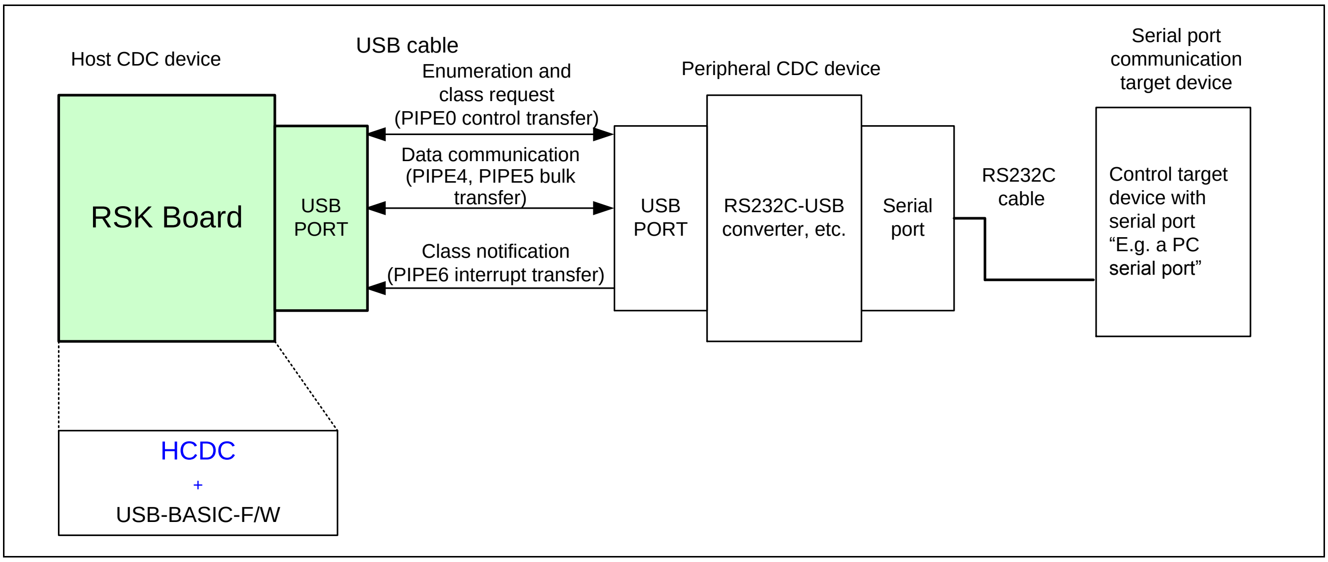

Figure 5.1 Example Operating Environment

5.2 Application Program Processing

The following lists application operation with respect to Figure 5.2 and Figure 5.5.

- CDC device attachment. (Corresponding to Process No.0-1)
- The connected CDC device is automatically initialized. (Process 2-1)
  Set RTS and DTR by using class request SET_CONTROL_LINE_STATE.
  To set communication speed, number of data bits, number of stop bits, and parity bit settings, use class request SET_LINE_CODING.
  To get communication speed, number of data bits, number of stop bits, and parity bit settings, use class request GET_LINE_CODING.
- Data Communication Start. (Process 2-2)
  Register a callback function to receive a report of the UART state. (Start Interrupt-IN transfer)
  From start of data receptionto completion (Bulk-IN forwarding start and callback generation)
- Change the baud rate of the connected device. (Process 4-1)
  If switch 2 is pressed, a new baud rate is selected. (Process 5-1)
  If switch 3 is pressed, the selected baud rate is activated. (Process5-2)
  And then transmit the SET_LINE_CODING request to the connected device.
- Data reception is completed. (Process 4-2)
  Communication with a USB device is completed when the callback function is generated from HCDC.
- Data transmission start. (Process 6-1)
  The received data is transmitted to the CDC device (loop back), and the data transmission is completed. (Bulk-OUT forwarding begins and the callback is generated)

- Data transmission is completed. (Process 7-2)
  Data reception is restarted. (Bulk-IN forwarding begins)
- Serial status reception is completed. (Process 4-3)
  Communication with a USB device is completed when the callback function(*usb_hcdc_smp_SerialStateReceive*) is generated from HCDC.
- Note: When data reception or data transmission fails, data reception is restarted.

5.3 Endpoint Specifications

The endpoints use by HCDC are shown in Table 5-1.

Table 5-1 Endpoint Specifications

Endpoint Number	Pipe Number	Transfer Method	Description
0	0	Control In/Out	Standard request, class request
Follows received Descriptor*1	4 or 5	Bulk In	Data transfer from device to host
	4 or 5	Bulk Out	Data transfer from host to device
	6	Interrupt In*2	State notification from device to host

Note)
*1 The Endpoint numbers are determined by the device's endpoint descriptors.
*2 The CDC device of the vendor class is connected, the interrupt transfer is not executed.

5.4 Connected CDC Peripheral

Please confirm the characteristics of the CDC peripheral before attempting to use it. When using a commercial USB-serial converter together with the CDC peripheral, check that the interface class code in the interface descriptor is "communication interface class" and not Vendor class. If it is, the CDC converter **will not work**.

If the USB serial converter is Vendor class, the following changes are necessary.

File: *r_usb_class_usrcfg.h*

```
/**********************************************
Macro definitions (USER DEFINE)
**********************************************/
/* Select CDC Interface class */
//#define USB_HCDC_IF_CLASS  USB_IFCLS_VEN        /* CDC Device Vendor class */
#define USB_HCDC_IF_CLASS    USB_IFCLS_CDCC       /* CDC Device CDC class */
```

5.5 List of APL Functions

Table 5-2 lists the functions of the sample application.

Table 5-2 List of Functions of Sample Application

Function Name	Description
Main	Main loop processing
usb_hcdc_main_init	System initialization Task start up processing for Host USB
usb_hcdc_main_task	HCDC sample application task
usb_hcdc_registration	HCDC driver registration
usb_hsmpl_class_check	Check connected device
usb_hsmpl_device_state	HCDC sample application status change callback function
usb_hcdc_smp_SendEncapsulatedCommand	Send class request : *SendEncapsulatedCommand*
usb_hcdc_smp_GetEncapsulatedResponse	Send class request : *GetEncapsulatedResponse*
usb_hcdc_smp_SetCommFeature	Send class request : *SetCommFeature*
usb_hcdc_smp_GetCommFeature	Send class request : *GetCommFeature*
usb_hcdc_smp_ClrCommFeature	Send class request : *ClearCommFeature*
usb_hcdc_smp_SetLineCoding	Send class request : *SetLineCoding*
usb_hcdc_smp_GetLineCoding	Send class request : *GetLineCoding*
usb_hcdc_smp_SendBreak	Send class request : *SendBreak*
usb_hcdc_smp_SetControlLineState	Send class request : *SetControlLineState*
usb_hcdc_smp_SerialStateReceive	Call-back function at Interrupt-IN notification
usb_hcdc_smp_InTransResult	Call-back function at Bulk-IN transaction end
usb_hcdc_smp_OutTransResult	Call-back function at Bulk-OUT transaction end
usb_hcdc_smp_crass_request_result	Call-back function at Send class request
usb_hcdc_smp_init	Function that transmits initialization request message to sample application
usb_hcdc_sw_request	Send switch check request
usb_hcdc_sw_process	Processing for pressed switch
usb_hcdc_get_line_coding_rcv_process	Processing received data from device after a *GetLineCoding* request above
usb_hcdc_smpl_message_send	Transfer message to mail box of demo sample application
usb_hsmpl_transfer_result	Transfer message to mail box of sample application by the transfer end
usb_hsmpl_dummy_fnc	Call-back dummy function

5.6 Host Application Task Sequence

The following explains how the LCD display is updated, control of state transitions, and other operations.

5.6.1 State Transitions

Figure 5.2 shows the application state transition. Each block is a program "state".

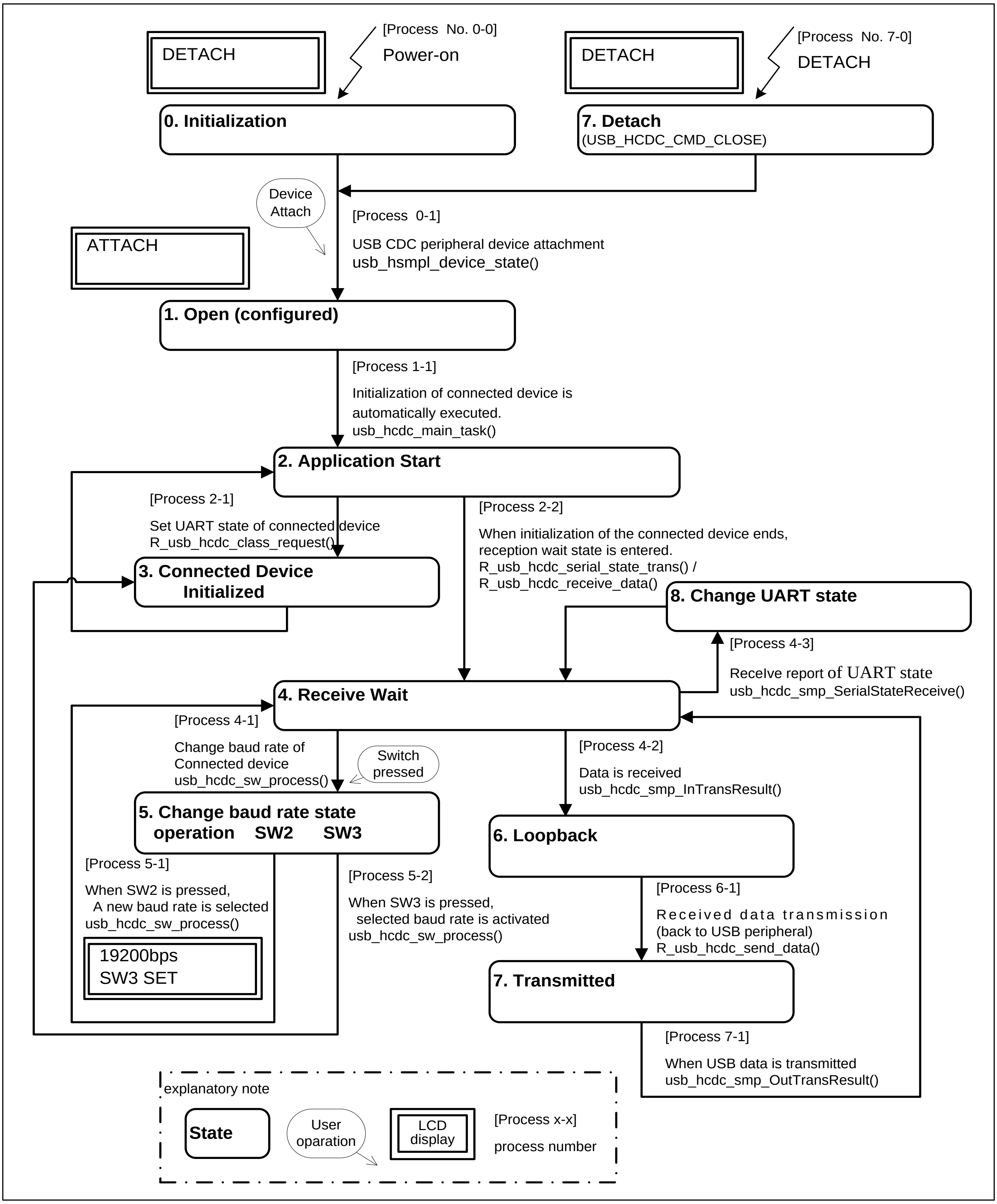

Figure 5.2 Application State Transitions

5.7 Processing Flow Graphs

The following shows the application task processing flow overview. Refer to the Table 5-3 for details on command processing.

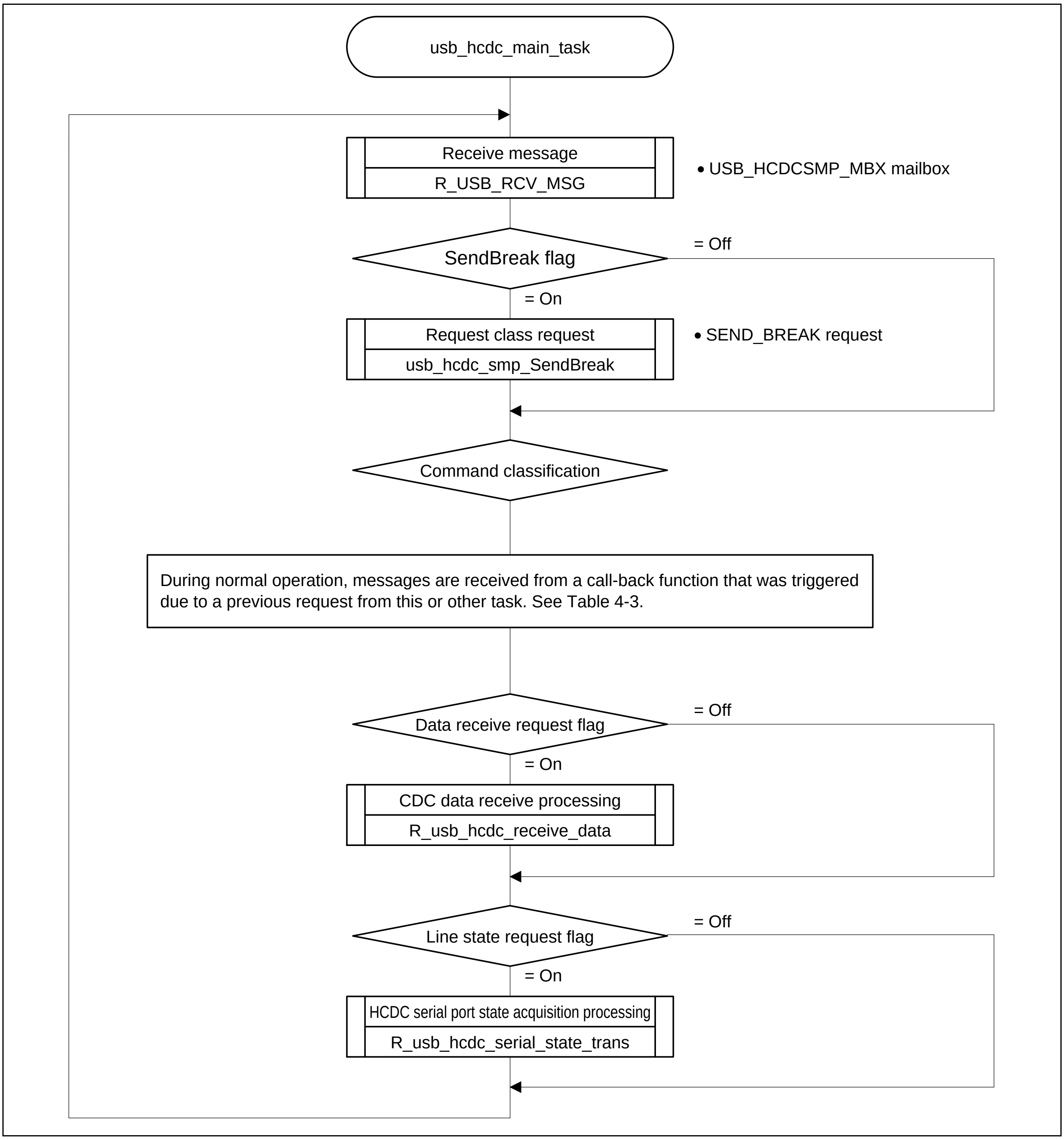

Figure 5.3 Outline of Host Application Task Processing Sequence

Table 5-3 Host Application Task Processing Details

Signal	Meaning	Remarks
USB_HCDC_CMD_INIT	Application initialization	Display application title. Ask the HCDC task to send class request *SetControlLineState* via *usb_hcdc_smp_SetControlLineState* ().
USB_HCDC_CMD_ SET_CONTROL_LINE_STATE	Class request *SetControlLineState* transfer completed	Ask the HCDC task to send class request *SetLineCoding* via *usb_hcdc_smp_SetLineCoding ()*.
USB_HCDC_CMD_ SET_LINE_CODING	Class request *SetLineCoding* transfer completed.	Ask HCDC to send class request ***G****etLineCoding* via *usb_hcdc_smp_GetLineCoding* ()
USB_HCDC_CMD_ GET_LINE_CODING	Class request *GetLineCoding* transfer completed.	Display acquired communication conditions. Set data receive request flag to ON. Set *SetLineState* request flag to ON.
USB_HCDC_CMD_RX_OK	Acquire receive data (receive length > 0)	Loopback transmit using *usb_hcdc_smp_OutTransResult ()*.
USB_HCDC_CMD_RX_NG	Did not receive any data	Set data receive request flag to ON. (Default demo is keep trying to receive.)
USB_HCDC_CMD_TX_OK **USB_HCDC_CMD_TX_NG**	Data transmit ended. Data transmit failed.	Set data receive request flag to ON. (Default demo is keep trying to receive.)
USB_HCDC_CMD_ RCV_SERIAL_STATE	Received class notification of *SerialState.*	Display line state. Set *SetLineState* request flag to ON.
USB_HCDC_CMD_ RCV_SERIAL_STATE_NG	Received class notification of *SerialState* failure.	Set *SetLineState* request flag to ON.
USB_HCDC_CMD_ SEND_BREAK	Class request *SendBreak* transfer completed.	Display *SendBreak* end message.
USB_HCDC_CMD_ SET_COMM_FEATURE	Class request *SetCommFeature* transfer completed.	Display *SetCommFeature* end message.
USB_HCDC_CMD_ GET_COMM_FEATURE	Class request *GetCommFeature* transfer completed.	Display *GetCommFeature* end message.
USB_HCDC_CMD_ CLR_COMM_FEATURE	Class request *ClearCommFeature* transfer completed.	Display *ClearCommFeature* end message.
USB_HCDC_CMD_ SEND_ENCAPSULATED_COMMAND	Class request *SendEncapsulatedCommand* transfer completed.	Display *SendEncapsulatedCommand* end message.
USB_HCDC_CMD_ GET_ENCAPSULATED_RESPONSE	Class request *GetEncapsulatedResponse* transfer completed.	Display *GetEncapsulatedResponse* end message.
USB_HCDC_CMD_SW_CHECK	Periodic processing. Select baud rate by user switch input.	SW2: The temporary baud rate is updated. Send USB_HCDC_CMD_SW_CHECK message to APL task. SW3: The baud rate is fixed and send USB_HCDC_CMD _SET_CONTROL_LINE_STATE message to APL task.
Undefined signal		Display received command.

5.8 Sequences charts APL-HCDC-HCD

The operation sequence of the sample application program is described below.

5.8.1 Startup to CDC Device Attachment

The sequence from sample application program startup through completion of enumeration, application task startup, and completion of pipe control register setting is illustrated in Figure 5.4 and Figure 5.5

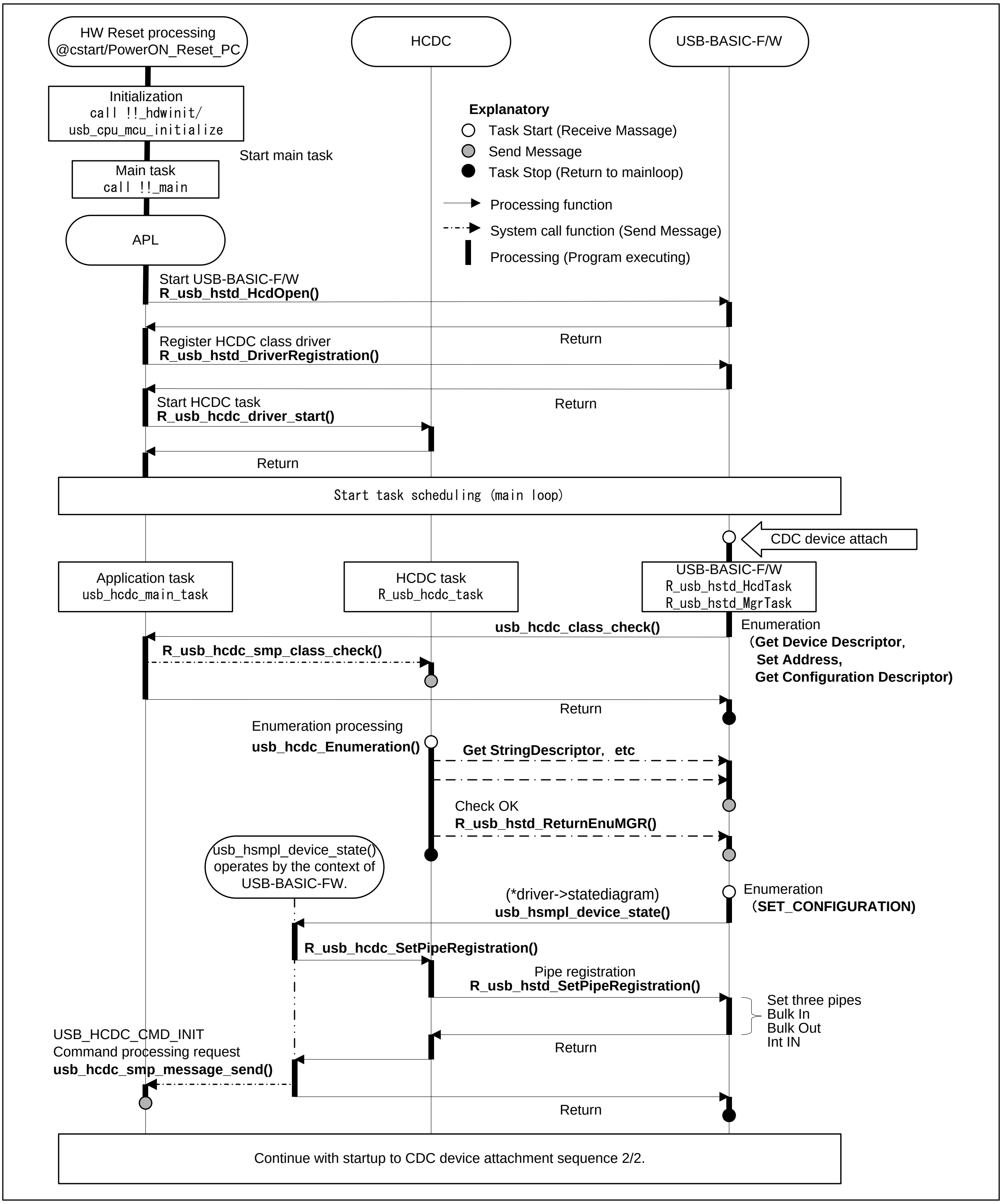

Figure 5.4 Startup to CDC Device Attachment Sequence (1/2)

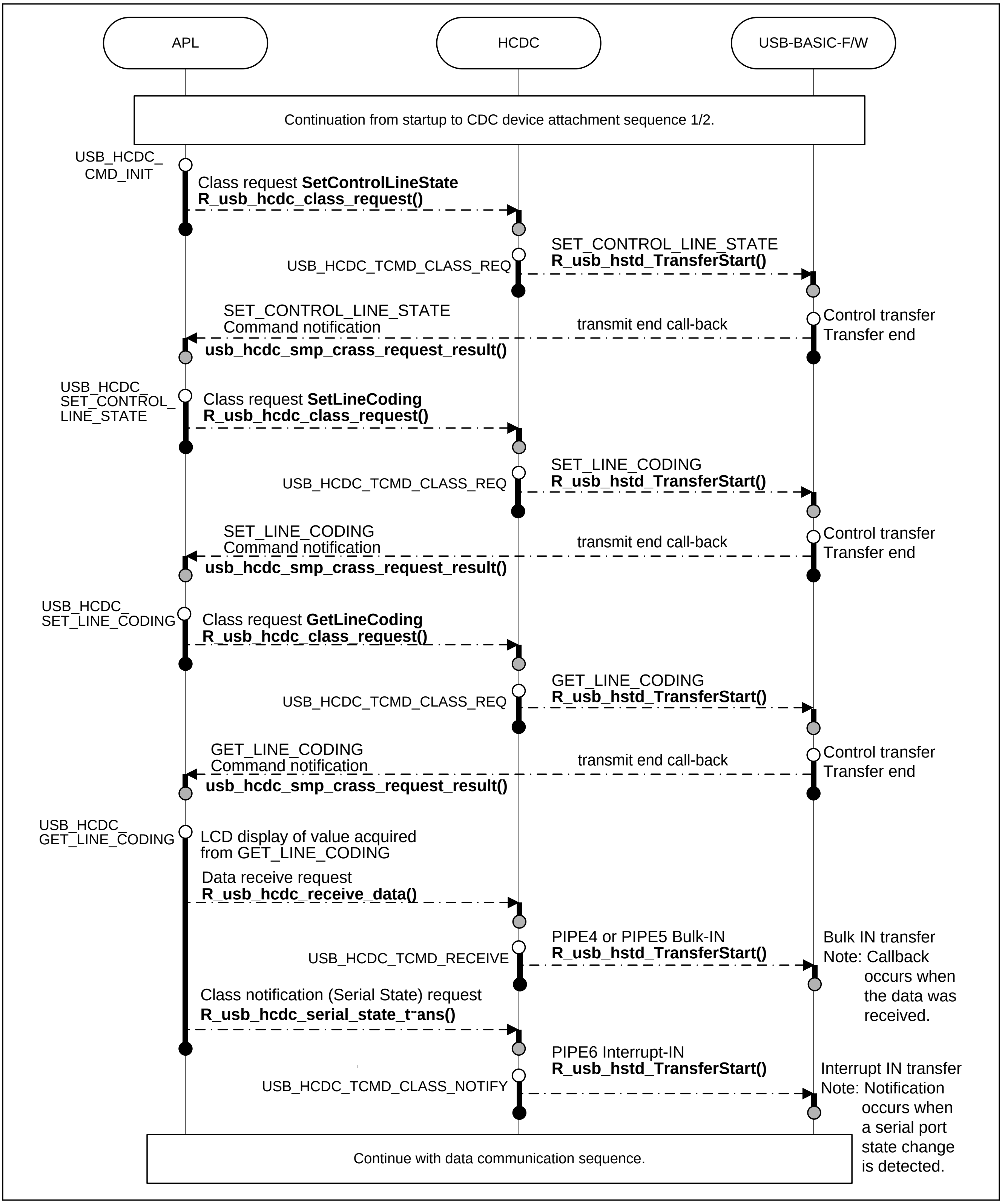

Figure 5.5 Startup to CDC Device Attachment Sequence (2/2)

5.8.2 Data Communication

The data transfer sequence is illustrated in Figure 5.6.

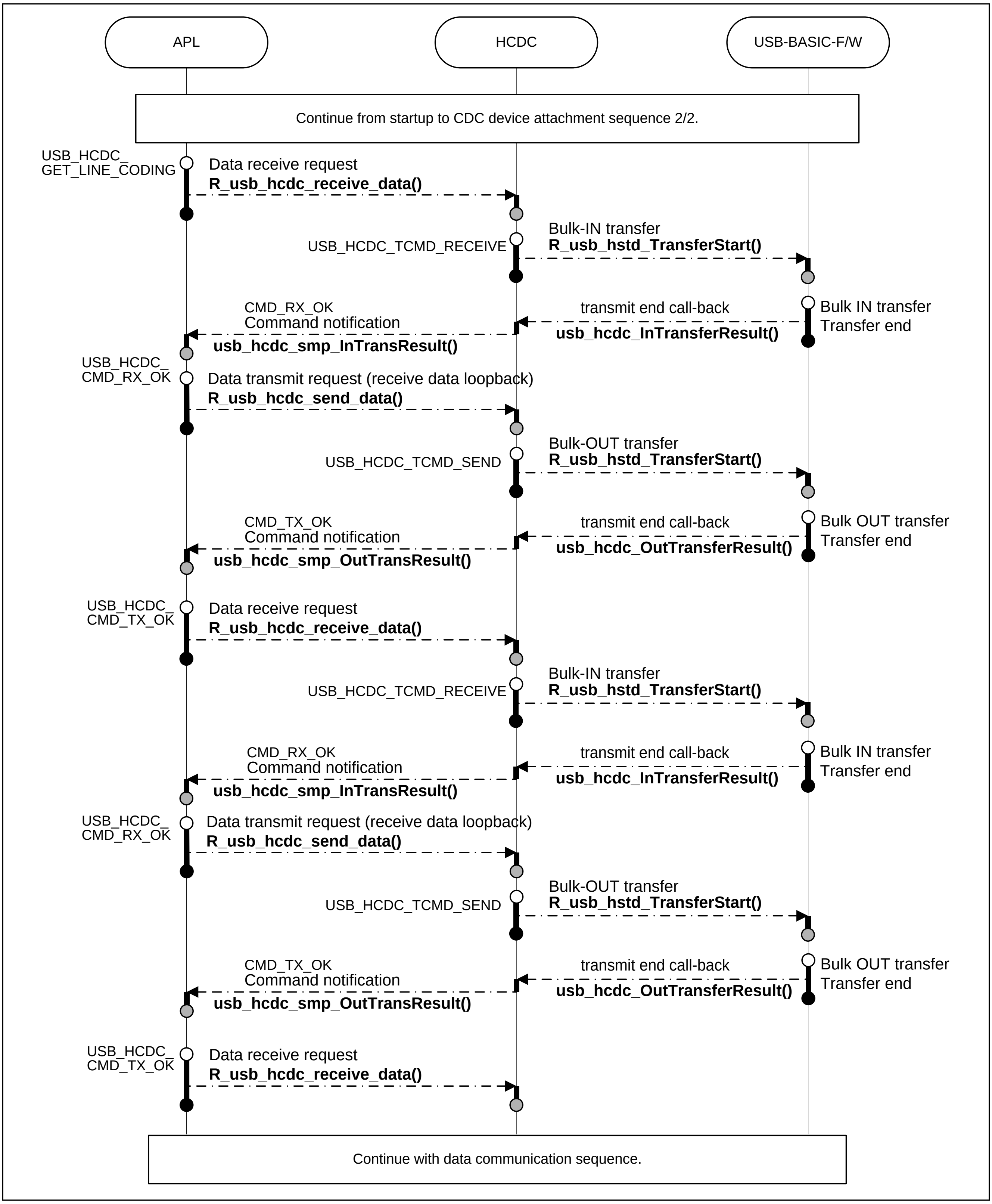

Figure 5.6 Data Communication Sequence

5.8.3 Serial Port State Change Notification

The serial port state change notification sequence is illustrated in Figure 5.7.

In order to receive a serial port state change, first perform the CDC notification *SerialState* receive processing.

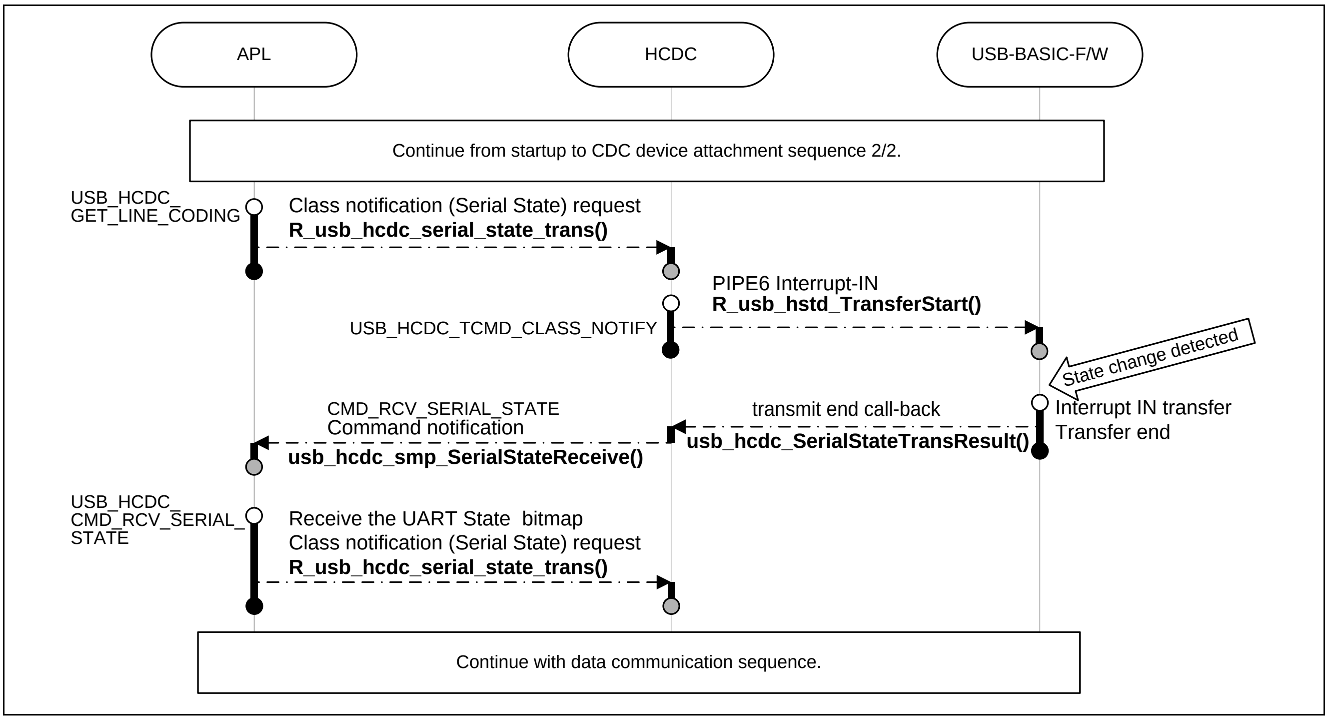

Figure 5.7 Serial Port State Change Notification Sequence

5.8.4 BREAK Signal Output

The BREAK signal output is illustrated in Figure 5.8.

The BREAK signal is demanded to the connected CDC peripheral device by the SEND_BREAK request when the global variable *usb_shcdc_test_send_break* is setting by USB_ON.

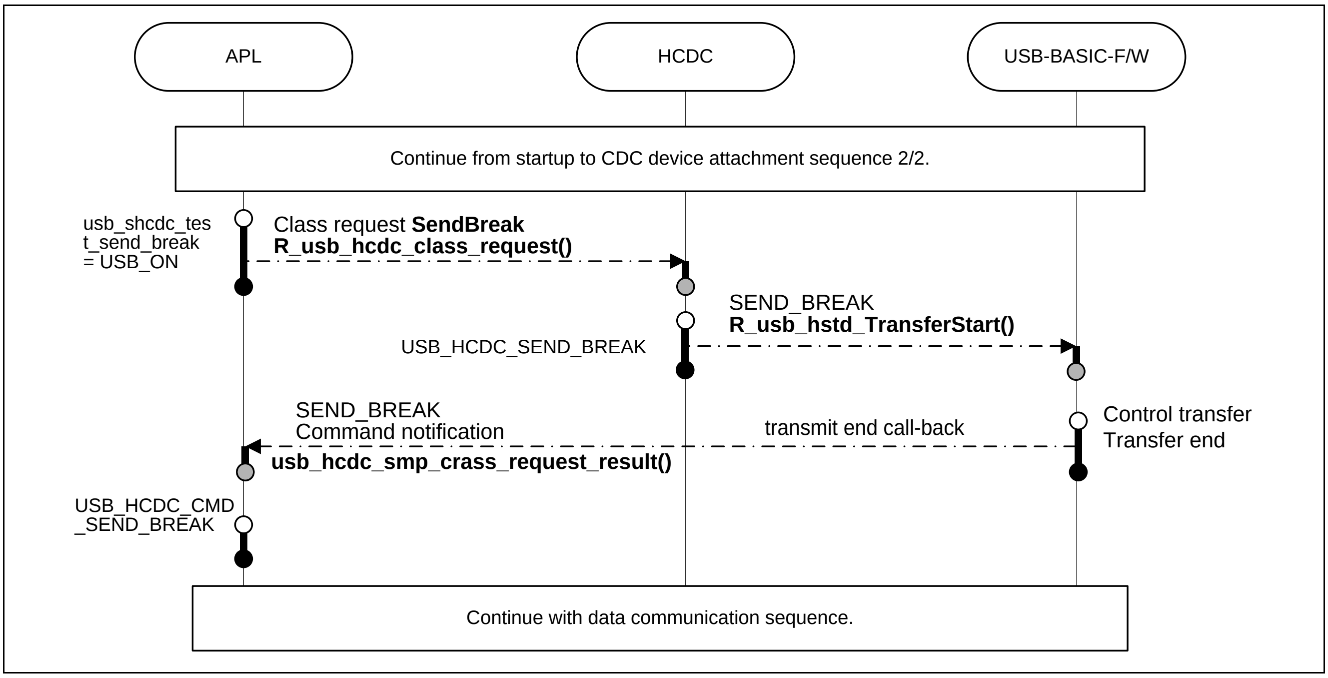

Figure 5.8 Break Signal Output Sequence

5.8.5 CDC Device Detach

The sequence when the CDC device is detached is illustrated in Figure 5.9.

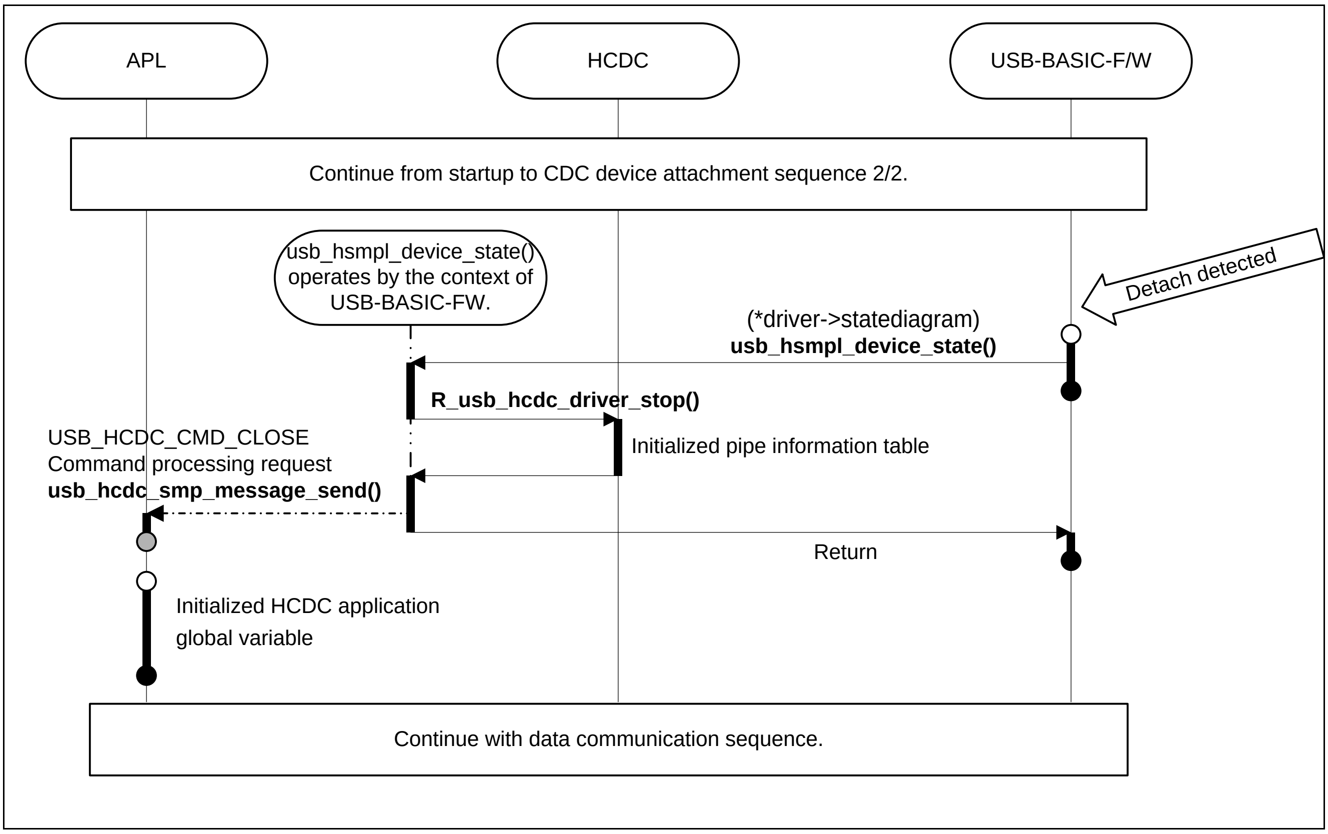

Figure 5.9 CDC Device Detach Sequence

6. Communication Device Class (CDC), PSTN, and ACM

This software conforms to the Abstract Control Model (ACM) subclass of the Communication Device Class specification, as specified in detail in the PSTN Subclass document listed in Chapter 1.2.

The Abstract Control Model subclass is a technology that bridges the gap between USB devices and earlier modems (employing RS-232C connections), enabling use of application programs designed for older modems, but also, for new applications that need only bulk transfer - large amounts of non time critical data – the CDC ACM model may be the most straightforward class solution.

6.1 Basic Functions

The main functions of HCDC are as follows.

1. Verify connected devices
2. Make communication line settings
3. Acquire the communication line state
4. Transfer data to and from the CDC peripheral device

6.2 Abstract Control Model Class Requests (Host to Device)

The software supports the following ACM class requests.

Table 6-1 CDC Requests

Request	Code	Description	Support
SendEncapsulatedCommand	0x00	Transmits AT commands, etc., as defined by the ACM protocol.	Yes
GetEncapsulatedResponse	0x01	Requests a response to a command transmitted by SendEncapsulatedCommand.	Yes
SetCommFeature	0x02	Enables or disables features such as device-specific 2-byte code and country setting.	Yes
GetCommFeature	0x03	Acquires the enabled/disabled state of features such as device-specific 2-byte code and country setting.	Yes
ClearCommFeature	0x04	Restores the default enabled/disabled settings of features such as device-specific 2-byte code and country setting.	Yes
SetLineCoding	0x20	Makes communication line settings (communication speed, data length, parity bit, and stop bit length).	Yes
GetLineCoding	0x21	Acquires the communication line setting state.	Yes
SetControlLineState	0x22	Makes communication line control signal (RTS, DTR) settings.	Yes
SendBreak	0x23	Transmits a break signal.	Yes

For details concerning the Management Element Requests, refer to Table 19, “Class-Specific Request Codes” in “USB Class Definitions for Communications Devices”, Revision 1.2, and the Abstract Control Model requests, refer to Table 11, “Requests - Abstract Control Model” in “USB Communications Class Subclass Specification for PSTN Devices”, Revision 1.2.

6.3 CDC Notifications (Notifications from Device to Host)

The CDC notifications supported and not supported by the software are shown by Table 6-2.

Table 6-2 CDC Notifications

Notification	Code	Description	Supported
NetworkConnection	0x00	Notification of network connection state	No
ResponseAvailable	0x01	Response to GET_ENCAPSLATED_RESPONSE	No
SerialState	0x20	Notification of serial line state	Yes

For details concerning the Management Element Requests, refer to Table 20, "Class-Specific Notification Codes" in "USB Class Definitions for Communications Devices", Revision 1.2, and the Abstract Control Model requests, refer to Table 28, "Requests - Abstract Control Model" in "USB Communications Class Subclass Specification for PSTN Devices", Revision 1.2.

7. USB Host Communication Device Class Driver (HCDC)

7.1 Basic Functions

This software conforms to the Abstract Control Model subclass of the communication device class specification. See Chapter 1.2 item 2 and 3.

The main functions of HCDC are to:

1. Send class requests to the CDC peripheral
2. Transfer data to and from the CDC peripheral
3. Receive communication error information from the CDC peripheral

7.2 HCDC Task Description

This task receives messages to mailbox USB_HCDC_MBX and performs processing according to the message. See Table 7-1.

Table 7-1 Processing according to Received HCDC Message Type

Message	Processing	Message Source
USB_HCDC_ TCMD_OPEN	Gets the string descriptor and sets the pipe according to the enumeration sequence.	*usb_hsmp_class_check()* *usb_hcdc_CheckResult()* Using the callback functions above, USB-BASIC-F/W and HCDC check for correct operation of the connected device during enumeration.
USB_HCDC_ TCMD_RECEIVE	"A Bulk-IN transfer is started. The application is later notified when the data transfer is completed."	When a Bulk-IN transfer is complete the API function *R_usb_hcdc_receive_data()* is executed.
USB_HCDC_ TCMD_SEND	"A Bulk-OUT transfer is started. The application is later notified when the data transfer is completed."	When Bulk-OUT transfer is completed the API function *R_usb_hcdc_send_data()* is executed.
USB_HCDC_TCMD_ CLASS_NOTIFY	Start Interrupt-IN transfer. The application is later notified when the data transfer is completed.	When an Interrupt-IN transfer is completed, the API function *R_usb_hcdc_serial_state_trans()* is executed.
USB_HCDC_TCMD_ CLASS_REQ	A CDC class request is issued as specified by application argument. The application is notified when the control transfer is completed.	The API function *ming*is executed .

7.3 Target Peripheral List (TPL)

Ahost, conforming to a class specification, is not required to support operation with all types of USB peripherals. It is up to the manufacturer of the host to determine what peripherals the device will support, and to provide a list of those peripherals. This is the called the "Target Peripheral List (TPL)".

TPL consists of VIDs and PIDs . If all VIDs (/PIDs) are to be ignored, USB_NOVENDOR (/USB_NOPRODUCT) is written to TPL. Please refer to the *usb_hcdc_Enumaration()* function in the *r_usb_hcdc_driver.c* file for TPL.

7.4 Structures

7.4.1 HCDC Request Structure

Table 7-2 describes the "UART settings" parameter structure used for the CDC requests *SetLineCoding* and *GetLineCoding*.

Table 7-2 USB_HCDC_LineCoding_t Structure

Type	Member	Description	Remarks
uint32_t	dwDTERate	Line speed	Unit: bps
uint8_t	bCharFormat	Stop bits setting	
uint8_t	bParityType	Parity setting	
uint8_t	bDataBits	Data bit length	

Table 7-3 describes the "UART settings" parameter structure used for the CDC requests *SetControlLineState*.

Table 7-3 USB_HCDC_ControlLineState_t Structure

Type	Member	Description	Remarks
uint16_t (D15-D8)	rsv1:8	Reserved1	
uint16_t (D7-D2)	rsv2:6	Reserved2	
uint16_t (D1)	bRTS:1	Carrier control for half duplex modems 0 - Deactivate carrier, 1 - Activate carrier	
uint16_t (D0)	bDTR:1	Indicates to DCE if DTE is present or not 0 - Not Present, 1 - Present	

Table 7-4 describes the "AT command" parameter structure used for the CDC requests *SendEncapsulatedCommand* and *GetEncapsulatedResponse*.

Table 7-4 USB_HCDC_Encapsulated_t Structure

Type	Member	Description	Remarks
uint8_t	*p_data	Area where AT command data is stored	
uint16_t	wLength	Size of AT command data	Unit: byte

Table 7-5 describes the "Break signal" parameter structure used for the CDC requests *SendBreak*.

Table 7-5 USB_HCDC_BreakDuration_t Structure

Type	Member	Description	Remarks
uint16_t	wTime_ms	Duration of Break	Unit: ms

7.4.2 CommFeature Function Selection Union

Table 7-6 and Table 7-7 describe the "Feature Selector" parameter structure used for the CDC requests *SetCommFeature* and *GetCommFeature*, and Table 7-8 describes the parameter union.

Table 7-6 USB_HCDC_AbstractState_t Structure

Type	Member	Description	Remarks
uint16_t	rsv1:8	Reserved1	
uint16_t	rsv2:6	Reserved2	
uint16_t	bDMS:1	Data Multiplexed State	
iomt16_t	bIS:1	Idle Setting	

Table 7-7 USB_HCDC_CountrySetting_t Structure

Type	Member	Description	Remarks
uint16_t	country_code	Country code in hexadecimal format as defined in [ISO3166],	

Table 7-8 USB_HCDC_CommFeature_t Structure

Type		Member	Description	Remarks
union	uint16_t	data	Status data	
	USB_HCDC_AbstractState_t	AbstractState	Abstract Control Model selection parameters. Refer to Table 7-6	
	USB_HCDC_CountrySetting_t	CountrySetting	Country Setting selection parameters. Refer to Table 7-7	
uint16_t		wValue	Feature selector code	

7.4.3 CDC Request Input Parameter Union

Table 7-9 describes the common parameter structure for CDC requests.

Table 7-9 USB_HCDC_ClassRequestParm_t Structure

Request	Request code Structure type	Member name	Description
SetLineCoding	USB_HCDC_SET_LINE_CODING / USB_HCDC_LineCoding_t	LineCoding	Data address send and receive in data stage. Refer to Table 7-2
GetLineCoding	USB_HCDC_GET_LINE_CODING / USB_HCDC_LineCoding_t		
SetControlState	USB_HCDC_SET_ CONTROL_LINE_STATE / USB_HCDC_ControlLineState_t	ControlLineState	Value set to the wValue field. Refer to Table 7-3
SendEncapsulated Command	USB_HCDC_SEND_ ENCAPSULATED_COMMAND / USB_HCDC_Encapsulated_t	Encapsulated	Data address send and receive in data stage, and value set to the wValue field. Refer to Table 7-4
GetEncapsulated Response	USB_HCDC_GET_ ENACAPSULATED_RESPONSE / USB_HCDC_Encapsulated_t		
SendBreak	USB_HCDC_SEND_BREAK / USB_HCDC_BreakDuration_t	BreakDuration	Value set to the wValue field. Refer to Table 7-5
SetCommFeature	USB_HCDC_SET_ COMM_FEATURE / USB_HCDC_CommFeature_t	*CommFeature	Data address send and receive in data stage. Refer to Table 7-8
GetCommFeature	USB_HCDC_GET_ COMM_FEATURE / USB_HCDC_CommFeature_t		
ClearCommFeature	USB_HCDC_CLR_COMM_FEATURE No structure		

7.4.4 CDC Request API Function Structure

Table 7-10 describes the CDC request parameter structure.

Table 7-10 USB_HCDC_ClassRequest_UTR_t Structure

Type	Member	Description
usb_addr_t	devadr	Device address
uint8_t	bRequestCode	Class request code. Refer to Table 7-9
USB_CDC_ClassRequestParm_t	parm	Parameter setup value. Refer to Table 7-9
usb_cb_t	complete	Class request processing end call-back function

7.4.5 CDC Notification Format

Table 7-11 and Table 7-12 describe the data format of the CDC notification.

Table 7-11 Response_Available notification format

Type	Member	Description	Remarks
uint8_t	bmRequestType	0xA1	
uint8_t	bRequest	RESPONSE_AVAILABLE (0x01)	
uint16_t	wValue	0x0000	
uint16_t	wIndex	Interface	
uint16_t	wLength	0x0000	
uint8_t	Data	--	

Table 7-12 Serial_State notification format

Type	Member	Description	Remarks
uint8_t	bmRequestType	0xA1	
uint8_t	bRequest	SERIAL_STATE (0x20)	
uint16_t	wValue	0x0000	
uint16_t	wIndex	Interface	
uint16_t	wLength	0x0002	
uint16_t	Data	UART State bitmap	Refer to Table 7-13

The host is notified of the "*SerialState*" when a change in the UART port state is detected. Table 7-13 describes the structure of the UART State bitmap.

Table 7-13 USB_HCDC_SerialState_t Structure

Type	Member	Description	Remarks
uint16_t (D15-D8)	rsv1:8	Reserved1	
uint16_t (D7)	rsv2:1	Reserved2	
uint16_t (D6)	bOverRun:1	Overrun error detected	
uint16_t (D5)	bParity:1	Parity error detected	
uint16_t (D4)	bFraming:1	Framing error detected	
uint16_t (D3)	bRingSignal:1	Incoming signal (Ring signal) detected	
uint16_t (D2)	bBreak:1	Break signal detected	
uint16_t (D1)	bTxCarrier:1	Line connected and ready for communication	Data Set Ready
uint16_t (D0)	bRxCarrier:1	Carrier detected on line	Data Carrier Detect

h

7.5 List of HCDC API Functions

The HCDC API is shown in Table 7-14. The API functions are called by the application program.

Table 7-14 List of HCDC API Functions

Function	Description	Notes
R_usb_hcdc_task	HCDC task processing	
R_usb_hcdc_smp_class_check	Check for correct operation of connected device	
R_usb_hcdc_driver_start	HCDC driver task start	
R_usb_hcdc_driver_stop	HCDC driver task stop	
R_usb_hcdc_SetPipeRegistration	Pipe Information Table update and pipe setting processing	
R_usb_hcdc_receive_data	HCDC USB data reception request to HCDC	
R_usb_hcdc_receive_data_end	USB data receive termination request to HCDC	
R_usb_hcdc_send_data	request USB data transmission of HCDC	
R_usb_hcdc_send_data_end	USB data transfer termination request of HCDC.	
R_usb_hcdc_serial_state_trans	CDC notification request of HCDC	
R_usb_hcdc_serial_state_trans_stop	CDC notify termination request of HCDC	
R_usb_hcdc_device_information	The USB state of a connected device is acquired from HCDC	
R_usb_hcdc_change_device_state	The USB status change of the connected device is requested of HCDC	
R_usb_hcdc_class_request	Sends CDC request	

R_usb_hcdc_task

HCDC task processing

Format

void R_usb_hcdc_task(void)

Argument

— —

Return Value

— —

Description

Calls the *usb_hcdc_task()* function. The HCDC task processes requests from the application, and the result is notified back to the application.

Note

Please refer to the USB-BASIC-F/W application note about this program loop.

Example

```
void usb_apl_task_switch(void)
{
  while( 1 )
  {

    if( USB_FLGSET == R_usb_cstd_Scheduler() )    /* Scheduler */
    {
       R_usb_hstd_HcdTask();    /* HCD Task */
       R_usb_hstd_MgrTask();    /* MGR Task */
       usb_hcdc_main_task();    /* HCDC Application Task */
       R_usb_hcdc_task();       /* HCDC Task */

    }
    else
    {
    }
  }
}
```

R_usb_hcdc_smp_class_check

Check descriptor

Format

void R_usb_hcdc_smp_class_check (uint8_t **table)

Argument

**table	Address array of the device information
	[0] : Address of Device Descriptor
	[1] : Address of Configuration Descriptor
	[2] : Address of global variable that mean the Device Address

Return Value

—	—

Description

This function asks the HCDC task whether operation of the connected device has been succeeded or not. Call this function when the USB-BASIC-F/W executes the *classcheck* callback.

The HCDC task references the endpoint descriptor of the device's configuration descriptor, then edits the Pipe Information Table and checks the pipe information of the pipes to be used.

Note

Example

```
void  usb_hcdc_registration(void)
{
  usb_hcdreg_t  driver;

  driver.ifclass      = USB_IFCLS_CDCC;
  driver.classcheck   = &R_usb_hcdc_smp_class_check;
  driver.statediagram = &usb_hcdc_device_state;
  R_usb_hstd_DriverRegistration(&driver);
}
```

R_usb_hcdc_driver_start

HCDC driver start

Format

void R_usb_hcdc_driver_start(void)

Argument

— —

Return Value

— —

Description

This function starts the HCDC driver task.

Note

Example

```
void usb_hcdc_task_start( void )
{
  /* Target board initialize */
  usb_cpu_target_init();

  /* USB-IP initialized */
  R_usb_hstd_ChangeDeviceState(USB_DO_INITHWFUNCTION);

  /* HCD driver open & registration */
  R_usb_hstd_HcdOpen();              /* HCD task, MGR task open */
  usb_hcdc_registration();           /* HCDC driver registration */
  R_usb_hcdc_driver_start();         /* HCDC Task start */

  /* Scheduler initialized */
  R_usb_hstd_ChangeDeviceState(USB_DO_SETHWFUNCTION);

}
```

R_usb_hcdc_driver_stop

HCDC driver stop

Format

void R_usb_hcdc_driver_stop(void)

Argument

— —

Return Value

— —

Description

This function initializes the pipe information table.

Note

Example

```
USB_STATIC void usb_hcdc_device_state(uint16_t data, uint16_t state)
{
  switch( state )
  {
   case USB_STS_DETACH:
      usb_smpl_set_suspend_flag(USB_NO);
      usb_shcdc_line_speed_control_seq = USB_HCDC_SMP_LINE_SPEED_INIT;
      R_usb_hcdc_driver_stop();
      usb_hcdc_smp_message_send(USB_HCDC_CMD_CLOSE);
   break;
             ・
             ・
             ・
}
```

R_usb_hcdc_SetPipeRegistration

Pipe information table update and pipe setting processing

Format

void R_usb_hcdc_SetPipeRegistration (usb_addr_t devaddr)

Argument

devaddr Device address

Return Value

— —

Description

This function updates the address field of the Pipe Information Table, and selects the pipe used by the hardware for CDC communication.

A total of three pipes are set in HCDC: The Bulk IN and Bulk OUT pipes for data communications, as well as an Interrupt IN pipe for receiving the serial state.

Note

1. Refer to the USB-BASIC-F/W application note for information about the Pipe Information Table.
2. Please set another field in the Pipe Information Table beforehand by referring to the endpoint descriptor.

Example

```
void usb_hcdc_smp_open(usb_adrr_t devadr)
{
  usb_er_t      err;

  if (devadr != 0)
  {
    usb_shcdc_devadr = devadr;     /* Device Address store */

    /* Host CDC Pipe Registration */
    err = R_usb_hcdc_SetPipeRegistration(usb_shcdc_devadr);
  }
}
```

R_usb_hcdc_receive_data

Request data reception

Format

usb_er_t	R_usb_hcdc_receive_data (uint8_t *table, usb_leng_t size, usb_cb_t complete)

Argument

*table	Pointer to receive data buffer address
size	Reception size
complete	Process completion notice callback function

Return Value

USB_E_OK	Success
USB_E_ERROR	Failure, argument error

Description

This function requests USB data reception of USB-BASIC-F/W

"*size*" bytes is later received at the address given by argument "**table*".

When data reception is complete, (that is, data reception count is "*size*" bytes or there was a short packet reception, the callback function is executed.

Note

1. The data receive process results are obtained by the argument "*usb_utr_t **" of the callback function
2. Refer to the USB-BASIC-F/W application note for the Data Transfer structure *usb_utr_t*.

Example

```
/* Example of usage. */
 uint8_t    data[64];                               /* Data buff */
 usb_leng_t size = 64;                              /* Data size */

   R_usb_hcdc_receive_data((uint8_t *)data, size, (usb_cb_t)&usb_complete)

/* Callback function */
void  usb_complete( usb_utr_t *mess );
{
  /*  Describe the processing performed when the USB receive is completed.  */
}
```

R_usb_hcdc_receive_data_end

End of data reception

Format

usb_er_t R_usb_hcdc_receive_data_end (void)

Argument

— —

Return Value

USB_E_OK	Success
USB_E_ERROR	Failure, argument error
USB_E_QOVR	Overlap (transfer end request for the pipe during transfer end.)

Description

This function requests USB-BASIC-F/W to forcibly end a data receive transfer.

End of transfer is notified via the callback function that was specified when the data transfer was requested with *R_usb_hcdc_receive_data()*. The remaining data length of transmission and reception, pipe control register value, and transfer status = USB_DATA_STOP are found in the argument of the callback function (in the *usb_utr_t* structure).

Note

1. The data transmit process results are obtained by the argument "*usb_utr_t* *" of the callback function, see Description above .
2. Refer to the USB-BASIC-F/W application note for the Data Transfer structure *usb_utr_t*.

Example

```
void  usb_smp_task(void)
{

  /* reception end request */
  err = R_usb_hcdc_receive_data_end();

  return err;
    :
}
```

R_usb_hcdc_send_data

Request data transmission

Format

usb_er_t R_usb_hcdc_send_data(uint8_t *table, usb_leng_t size, usb_cb_t complete)

Argument

*table	Pointer to transmit data buffer address
size	Transfer size
complete	Process completion notice callback function

Return Value

USB_E_OK	Success
USB_E_ERROR	Failure, argument error

Description

This function requests a USB data transmission fo the USB-BASIC-F/W.

"*size*" bytes are transmitted from the address given by argument "*table*".

When the data transmit processing is complete, the callback function *complete* is called.

Note

1. Data transmission results are obtained by the argument "*usb_utr_t* *" of the callback function
2. Refer to the USB-BASIC-F/W application note forinformation ob the Data Transfer structure *usb_utr_t*.

Example

```
/* Example of usage. */
  uint8_t    data[] = {0x01,0x02,0x03,0x04,0x05}; /* USB send data */
  usb_leng_t size = 5;                             /* Data size */

  R_usb_hcdc_send_data((uint8_t *)data, size,(usb_cb_t)&usb_complete)

/* Callback function */
void  usb_complete( usb_utr_t *mess );
{
  /*  Describe the processing performed when the USB transmit is completed.  */
}
```

R_usb_hcdc_send_data_end

Terminate a data transmission transfer

Format

usb_er_t	R_usb_hcdc_send_data_end (void)

Argument

—	—

Return Value

USB_E_OK	Success
USB_E_ERROR	Failure, argument error
USB_E_QOVR	Overlap (transfer end request for the pipe during transfer end.)

Description

This function requests USB-BASIC-F/W to forcibly end a data transfer .

Transfer end is notified by the callback function which was given given when the data transfer was requested by *R_usb_hcdc_send_data*(). The remaining data length of the transmission, pipe control register value, and transfer status = USB_DATA_STOP are found in the argument of the callback function (in the *usb_utr_t* structure).

Note

1. The data transmit process results are obtained by the argument "*usb_utr_t* *" of the callback function. See above.
2. Refer to the USB-BASIC-F/W application note for information about the Data Transfer structure *usb_utr_t*.

Example

```
void  usb_smp_task(void)
{

  /* Transfer end request */
  err = R_usb_hcdc_send_data_end();

  return err;
    :
}
```

R_usb_hcdc_serial_state_trans

Request CDC notification

Format

usb_er_t R_usb_hcdc_serial_state_trans(usb_cb_t *complete)

Argument

complete Process completion notice callback function

Return Value

USB_E_OK	Success
USB_E_ERROR	Failure, argument error

Description

This function starts the reception of CDC notification (see Chapter 7.4.5).

The callback function *complete* is called after reception of CDC notification.

The argument of the callback function is of type *usb_utr_t**, a global variable of HCDC. Stored HCDC reception data is in the *tranadr* member of the *usb_utr_t** structure. The member "*result*" of this structure is USB_YES if the CDC notification was received correctly. The start address of "CDC Notification Format" (Refer to Table 7-12) area is set in the member (*tranadr*) of the argument which the callback function has. The serial state is found from "UART State bitmap" in this area.

Note

1. For information concerning the serial status bit pattern, refer to "Table 7-13".
2. The data transmit process results are obtained by the argument "*usb_utr_t **" of the callback function
3. Refer to the USB-BASIC-F/W application note for the Data Transfer structure *usb_utr_t*.
4. When the connected device is a vendor class device, the CDC notification is not demanded.
5. It doesn't correspond to the *ResponseAvailable* notification.

Example

```
void usb_hcdc_main_task(void)
{
  USB_MGRINFO_t          *mess;
  usb_er_t               err;

  while (1)
  {
  err  = R_USB_RCV_MSG(USB_HCDCSMP_MBX,(USB_MSG_t**)&mess);
  if (err == USB_E_OK)
  {
   err = R_usb_hcdc_serial_state_trans( mess,
          (usb_cb_t *)&usb_hcdc_smp_SerialStateReceive );
   if( err != USB_E_OK )
   {
      USB_PRINTF0("### usb_pcdc_MainTask function bulk read error\n");
   }
  }
}

/* Example of a callback function of R_usb_hcdc_serial_state_trans */
void usb_hcdc_smp_SerialStateReceive(usb_utr_t *mess)
{
  uint16_t *status;
  uint16_t msginfo;

  if (mess->result == USB_YES)
  {
   /* Command set */
   msginfo = USB_HCDC_CMD_RCV_SERIAL_STATE;
  }
  else
  {
   /* Command set */
   msginfo = USB_HCDC_CMD_RCV_SERIAL_STATE_NG;
  }
  status = (uint16_t *)mess->tranadr;      /* Status set */
  /* [0] bmRequestType/bRequest */
  /* [1] wValue */
  /* [2] wIndex */
  /* [3] wLength :2 */
  /* [4] data : Serial State(UART State bitmap) */
  usb_hcdc_smp_message_send( mess, status[4]);
}
```

R_usb_hcdc_serial_state_trans_end

CDC notification termination request

Format

usb_er_t	R_usb_hcdc_serial_state_trans_end (void)

Argument

—	—

Return Value

USB_E_OK	Success
USB_E_ERROR	Failure, argument error
USB_E_QOVR	Overlap (transfer end request for the pipe during transfer end.)

Description

This function requests USB-BASIC-F/W to forcibly end the data transfer.

The transfer end is notified via the callback function specified when the data transfer was requested *R_usb_hcdc_serial_satte_trans()*. The remaining data length of transmission and reception, pipe control register value, and transfer status = USB_DATA_STOP are found in the argument of the callback function (*usb_utr_t*).

Note

1. The data transmit process results are obtained by the argument "*usb_utr_t* *" of the callback function. See Description above.
2. Refer to the USB-BASIC-F/W application note for information on the Data Transfer structure *usb_utr_t*.

Example

```
void  usb_smp_task(void)
{

  /* notification end request */
  err = R_usb_hcdc_serial_state_trans_end();

  return err;
    :
}
```

R_usb_hcdc_class_request

Send a CDC class request

Format

usb_er_t R_usb_hcdc_class_request (USB_HCDC_ClassRequest_UTR_t *pram)

Argument

*pram CDC class request parameter.

Return Value

— Error code (USB_E_OK/USB_E_ERROR)

Description

The following CDC requests can be sent to the HCDC driver.

The request type is specified by the structure member *bRequestCode* of argument **parm*.

1. SendEncapsulatedCommand
2. GetEncapsulatedResponse
3. SetCommFeature
4. GetCommFeature
5. ClearCommFeature
6. SetLineCoding
7. GetLineCoding
8. SetControlLineState
9. SendBreak

Please refer to the sample application in *r_usb_hcdc_apl.c* for details on how to use this API.

Refer to Chapter 7.4.4 for the type *USB_HCDC_ClassRequest_UTR_t* structure of the argument.

Note

1. The class request transmission result is obtained by the argument "*usb_utr_t* *" of the callback function.
2. Refer to the USB-BASIC-F/W application note for the Data Transfer structure *usb_utr_t*.

Example

```
USB_HCDC_LineCoding_t          usb_shcdc_line_coding;
USB_HCDC_ClassRequest_UTR_t    utr_req;

/* Example of usage. */
  usb_shcdc_line_coding.dwDTERate    = USB_HCDC_SPEED_9600;
  usb_shcdc_line_coding.bDataBits    = USB_HCDC_DATA_BIT_8;
  usb_shcdc_line_coding.bCharFormat  = USB_HCDC_STOP_BIT_1;
  usb_shcdc_line_coding.bParityType  = USB_HCDC_PARITY_BIT_NONE;
  utr_req.bRequestCode          = USB_HCDC_SET_LINE_CODING;
  utr_req.complete              = (usb_cb_t)&usb_hcdc_smp_SetLine_CODING_Result;
  utr_req.parm.LineCoding       = &usb_shcdc_line_coding;
  utr_req.devadr                = devadr;
  return = R_usb_hcdc_class_request( utr_req );

/* Example of callback function. */
void usb_hcdc_smp_SetLine_CODING_Result(usb_utr_t *mess)
{
  /* Describe the processing performed when the class request is completed. */
}
```

R_usb_hcdc_device_information

Obtain USB device state information

Format

void R_usb_hcdc_device_information(uint16_t *deviceinfo)

Argument

*deviceinfo Table address to store the device information

Return Value

— —

Description

Obtain the USB device information. Store the following information to an address specified to the argument "*deviceinfo*".
[0]: Root port number (port 0: USB_0, port 1: USB_1)
[1]: USB state (unconnected: USB_STS_DETACH, enumerated: USB_STS_DEFAULT/USB_STS_ADDRESS, connected: USB_STS_CONFIGURED, suspended: USB_STS_SUSPEND)
[2]: Structure number (*g_usb_HcdDevInfo[g_usb_MgrDevAddr].config*)
[3]: Connection speed (FS: USB_FSCONNECT, LS: USB_LSCONNECT, unconnected: USB_NOCONNECT)

Notes

1. Provide 4 word area for the argument **deviceinfo*.
2. This function is called when the device address is 0, the following information is returned.
   (1) When there is not a device during enumeration (device is not connected).
   table[0] = USB_NOPORT, table[1] = USB_STS_DETACH
   (2) When there is a device during enumeration.
   table[0] = Port number, table[1] = USB_STS_DEFAULT

Example

```
void  usb_smp_task(void)
{
  uint16_t  tbl[4];
    :
  /* Device information check */
  R_usb_hcdc_device_information(tbl);
    :
}
```

R_usb_hcdc_change_device_state

USB device state change request

Format

usb_er_t R_usb_hcdc_change_device_state (usb_strct_t msginfo,
usb_strct_t keyword,
usb_cb_info_t complete)

Arguments

msginfo	USB state to be changed to.
keyword	Content varies depending on *msginfo*, for example port number.
complete	Callback function executed when the USB state has changed.

Return Value

USB_E_OK	Success
USB_E_ERROR	Failure, argument error

Description

Set the following value to argument *msginfo* to request to change the device state of the USB-BASIC-F/W.

- USB_DO_PORT_ENABLE / USB_DO_PORT_DISABLE
  Enable or disable a port specified by a keyword (on/off control of VBUS output).
- USB_DO_GLOBAL_SUSPEND
  Suspend a port specified by a keyword.
- USB_DO_GLOBAL_RESUME
  Resume a port specified by a keyword.
- USB_DO_CLEAR_STALL
  Cancel STALL of the device that uses a pipe specified by a keyword.

Notes

1. When a connection or disconnection is detected by the USB-BASIC-F/W, the enumeration sequence or the detach sequence is processed automatically by the USB-BASIC-F/W.
2. When transiting the USB state using this function, the USB state transition callback of the driver structure registered using the API function *R_usb_hstd_DriverRegistration()* is not called.

Example

```
void  usb_smp_task(void)
{
  R_usb_hcdc_change_device_state
   (USB_DO_GLOBAL_SUSPEND, USB_PORT0, usb_hsmpl_status_result);
}
```

8. Limitations

HDCD is subject to the following limitations.

1. The structures contain members of different types. (Depending on the compiler, this may cause address misalignment of structure members.)
2. Devices can connect to the HCDC, this is the only one. Please do not connect two or more devices simultaneously.

9. Setup for the e^2 studio project

(1). Start up e^2 studio.

* If starting up e^2 studio for the first time, the Workspace Launcher dialog box will appear first. Specify the folder which will store the project.

(2). Select [File] → [Import]; the import dialog box will appear.

(3). In the Import dialog box, select [Existing Projects into Workspace].

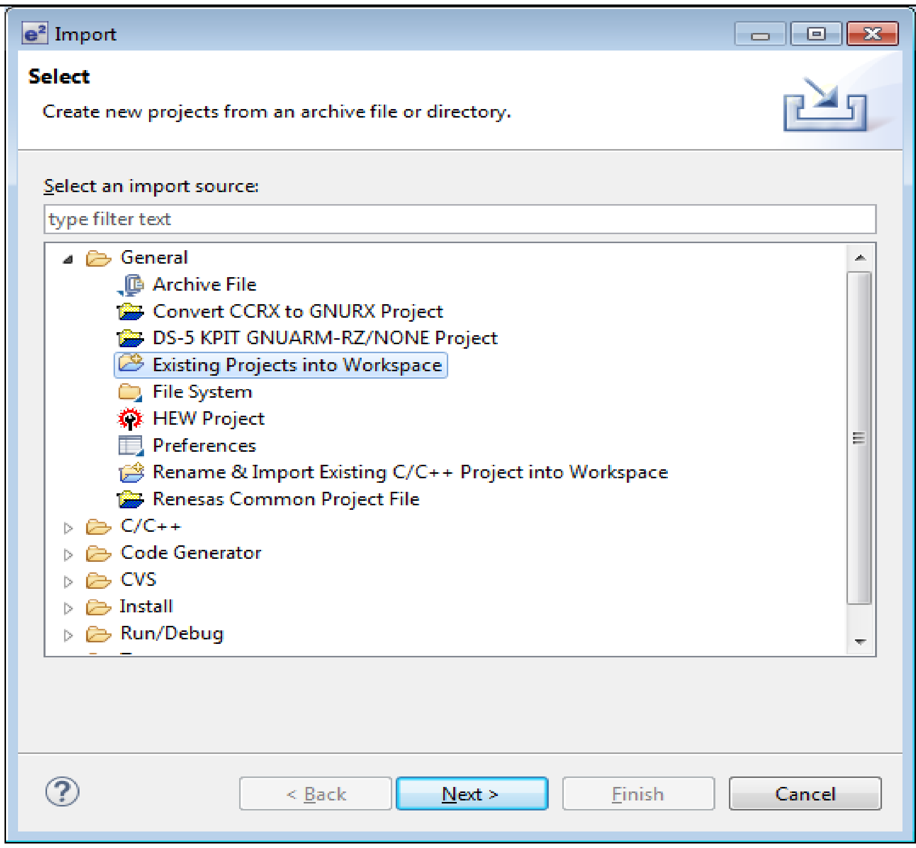

Figure 9-1 Select Import Source

(4). Press [Browse] for [Select root directory]. Select the folder in which [.cproject] (project file) is stored.

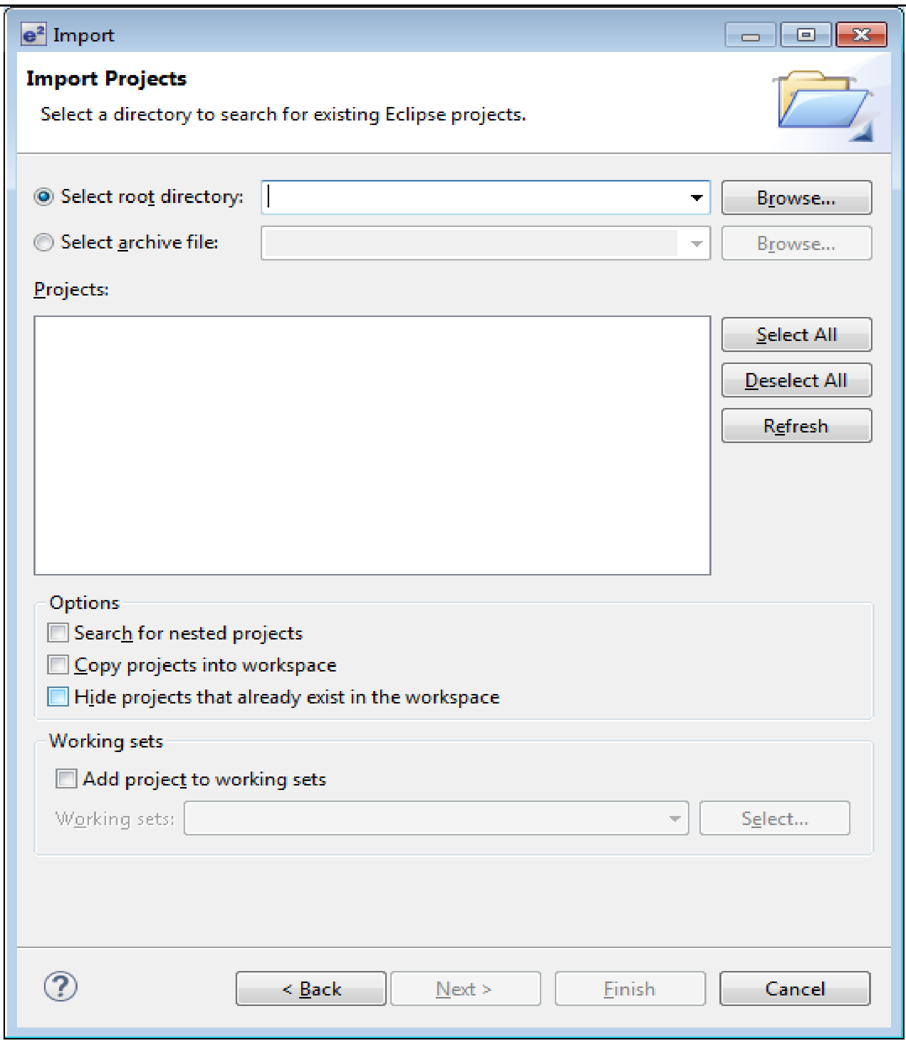

Figure 9-2 Project Import Dialog Box

(5). Click [Finish].

This completes the step for importing a project to the project workspace.

10. Using the e^2 studio project with CS+

This package contains a project only for e^2 studio. When you use this project with CS+, import the project to CS+ by following procedures.

Note:

The *rcpc* file is stored in "workspace\RL78\CCRL*devicename*" folder.

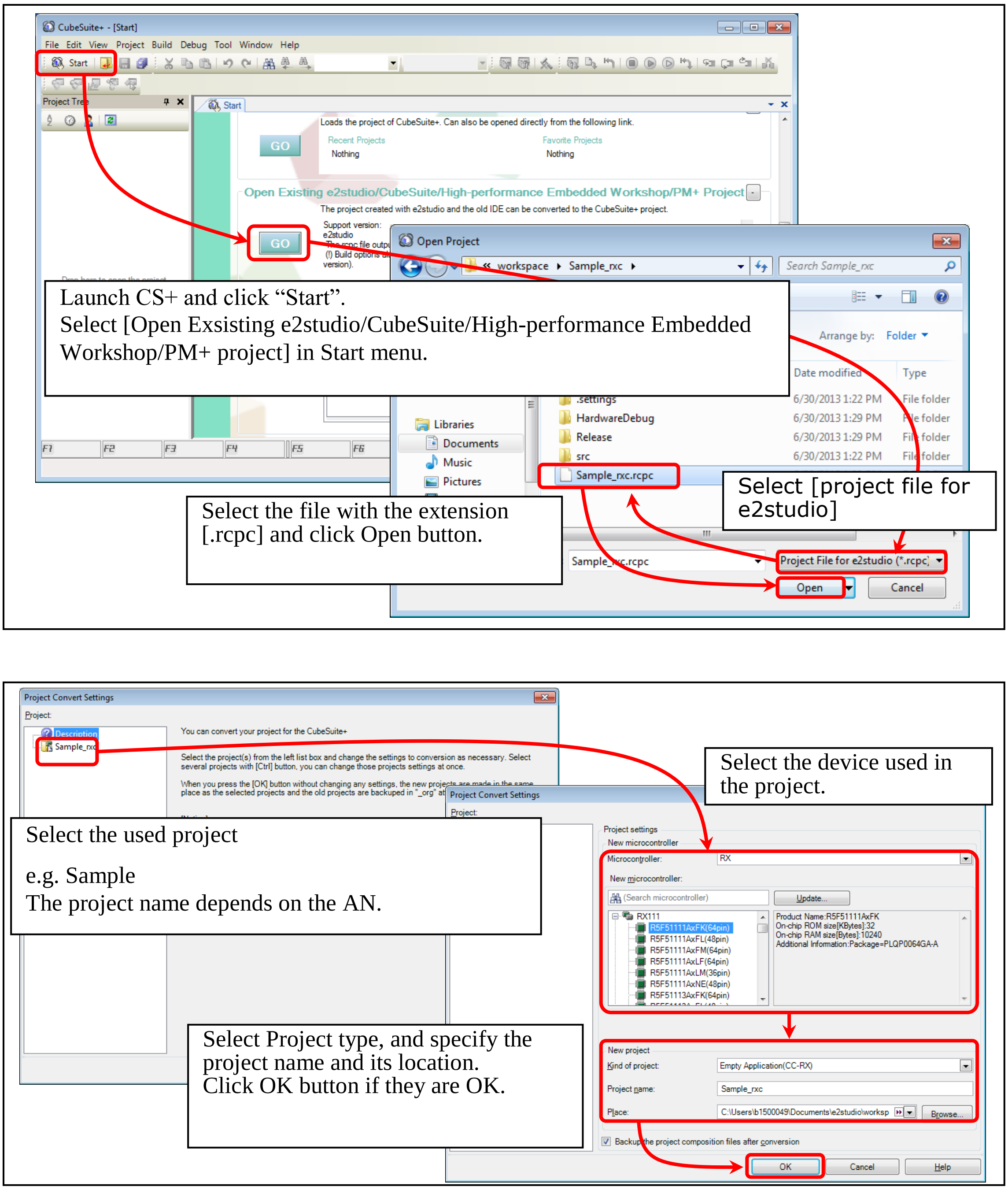

Figure 10-1 Using the e^2 studio project with CS+

Website and Support

Renesas Electronics Website
http://www.renesas.com/

Inquiries
http://www.renesas.com/contact/

Revision Record

Rev.	Date	Description: Page	Description: Summary
1.00	Apl.28.11	—	First edition issued
2.00	Nov. 30.12	—	Revision of the document by firmware upgrade
2.10	Aug. 01.13	—	RX111 is supported. Error is fixed.
2.11	Oct. 31. 13	—	1.4 Folder path fixed. 3.3.1 Folder Structure was corrected. Error is fixed.
2.12	Mar. 31.14	—	R8C is supported. Error is fixed.
2.13	Mar. 16. 15	—	RX111 is deleted from Target Device
2.14	Jan. 18. 16	—	Supported Technical Update (Document No. TN-RL*-A055A/E)
2.15	Mar. 28. 16	—	CC-RL compiler is supported.

General Precautions in the Handling of MPU/MCU Products

The following usage notes are applicable to all MPU/MCU products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Handling of Unused Pins

   Handle unused pins in accordance with the directions given under Handling of Unused Pins in the manual.

   — The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.

2. Processing at Power-on

   The state of the product is undefined at the moment when power is supplied.

   — The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the moment when power is supplied.
   In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the moment when power is supplied until the reset process is completed.
   In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the moment when power is supplied until the power reaches the level at which resetting has been specified.

3. Prohibition of Access to Reserved Addresses

   Access to reserved addresses is prohibited.

   — The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.

4. Clock Signals

   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has stabilized.

   — When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.

5. Differences between Products

   Before changing from one product to another, i.e. to a product with a different part number, confirm that the change will not lead to problems.

   — The characteristics of an MPU or MCU in the same group but having a different part number may differ in terms of the internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2801 Scott Boulevard Santa Clara, CA 95050-2549, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
9251 Yonge Street, Suite 8309 Richmond Hill, Ontario Canada L4C 9T3
Tel: +1-905-237-2004

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
Room 1709, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100191, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, P. R. China 200333
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 1207, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics India Pvt. Ltd.
No.777C, 100 Feet Road, HAL II Stage, Indiranagar, Bangalore, India
Tel: +91-80-67208700, Fax: +91-80-67208777

Renesas Electronics Korea Co., Ltd.
12F., 234 Teheran-ro, Gangnam-Gu, Seoul, 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 5.0