# Renesas USB MCU

R01AN0555EJ0215
Rev.2.15
Mar 28, 2016

USB Peripheral Communications Device Class Driver (PCDC) using USB Basic Mini Firmware

## Introduction

This document is an application note describing use of the USB Peripheral Communications Device Class Driver (PCDC) built using the USB Basic Mini Firmware.

## Target Device

RL78/G1C, RL78/L1C, R8C/34U, R8C/3MU, 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 document is an application note describing use of the USB peripheral Communication Device Class driver (PCDC) and communication port driver sample driver, using the USB Basic Mini FirmwareRenesas.

The ACM subclass of the CDC class is used. For USB applications that need bulk transfer - large amounts of non time critical data, this can be the most straightforward solution, since it can be used together with a PC UART terminal program. These host–side COM port terminal applications most often have a file transfer menu options built in. The terminal program will be opened towards the USB COM-port that will appear after enumeration.

## 1.1 Functions and Features

The PCDC conforms to the Abstract Control Model of the Communication Device Class specification (CDC) and enables communication with a USB Host.

This class driver is intended to be used in combination with the USB Basic Mini Firmware 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: HardwareUSB Peripheral Communications Device Class Driver (PCDC)
5. USB Basic Mini Firmware Application Note (Document No. R01AN0326EJ)
6. USB Peripheral Communications Device Class Driver (PCDC) Installation Guide for Basic Mini Firmware
7. FIT SCI Asynchronous Mode Module Application Note (Document No. R01AN1667EU)

Available from 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 |
| CDC | : | Communications Devices Class |
| CDCC | : | Communications Devices Class Communications Interface Class |
| CDCD | : | Communications Devices Class Data Class Interface |
| CPD | : | Serial Communication Port Driver |
| cstd | : | Prefix of function and file for Host & Peripheral USB-BASIC-F/W |
| Data Transfer | : | Generic name of Control transfer, Bulk transfer and Interrupt transfer |
| H/W | : | Renesas USB device |
| PCD | : | Peripheral control driver of USB-BASIC-F/W |
| PCDC | : | Communications Devices Class for peripheral |
| PCDCD | : | Peripheral Communications Devices Class Driver |
| PP | : | Pre-processed definition |
| pstd | : | Prefix of function and file for Peripheral USB-BASIC-F/W |
| RSK | : | Renesas Starter Kit |
| Scheduler | : | Used to schedule functions, like a simplified OS. |
| Scheduler Macro | : | Used to call a scheduler function |
| SCI | : | Serial Communication Interface |
| SW1/SW2/SW3 | : | Switch implemented on RSK board |
| Task | : | Processing unit |
| USB | : | Universal Serial Bus |
| USB-BASIC-F/W | : | USB Basic Mini Firmware for Renesas USB device |

## 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.

Chapter 5 explains how the sample application works. You will change this to create your own solution.

Understand how all code modules are 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 priority. Further, tasks are intended to be non-blocking by using a documented callback mechanism. The task mechanism is described in Chapter 1.2 above, "BASIC-FW Application Note".

All PCDC tasks are listed in Chapter 4.3 below.

## 2. How to Register Class Driver

The class driver which the user creates must be registered with the BASIC-FW. Please consult function *usb_papl_registration()* in *r_usb_pcdc_apl.c* on how to register a class driver with the BASIC-FW.

For details, please refer to the BASIC-FW 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. Renesas Starter Kit for RL78/L1C (Product No: R0K50110PC010BR)

c. R8C/34K Group USB Peripheral Evaluation Board (Product No: R0K5R8C34DK2PBR)

## 4. Software Configuration

### 4.1 Module Configuration

Figure 4-1 shows the configuration of the modules related to PCDC. Table 4.1 lists the software modules.

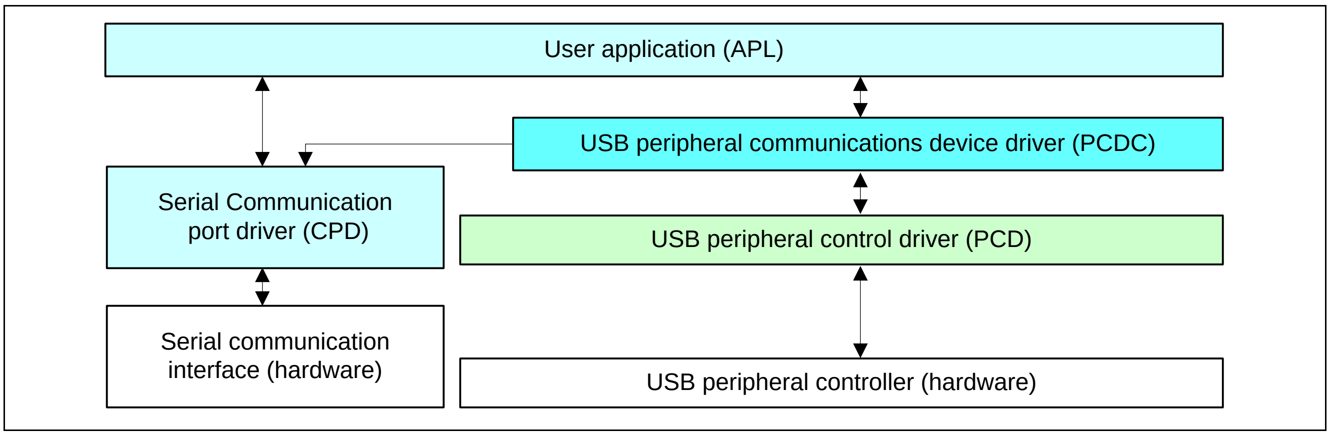

User application (APL)
USB peripheral communications device driver (PCDC)
Serial Communication port driver (CPD)
USB peripheral control driver (PCD)
Serial communication interface (hardware)
USB peripheral controller (hardware)

**Figure 4-1 Source Code Block Diagram**

**Table 4.1 Modules**

| Module | Description |
|---|---|
| APL | User application program. |
| PCDC | Sends requests from the APL for requests and data communication involving the CDC to the PCD. |
| PCD | USB peripheral hardware control driver. |
| CPD | Serial port control driver |

The user application (APL) and PCDC each run as tasks,called by the scheduler.

PCDC communicates with the host via PCD.

APL communicates over USB via PCDC and over the serial port via CPD.

## 4.2 Structure of Files and folders

### 4.2.1 Folder Structure

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

The source codes dependent on each device and evaluation board are stored in each hardware resource folder (\***devicename***\*src*\*HwResource*).

```
workspace
    +[ RL78 / R8C ]
      +[ CCRL / CS+ / IAR / e2 studio / HEW ]
        + [RL78G1C / RL78L1C / R8C3MK / R8C3MU / R8C34K / R8C34U ]
             + UART                                      UART build result
             + ECHO                                      ECHO build result
             + src
               +——— PCDC [ Communication Device  Class driver ]   See Table 4.2
               |          +——— inc                       Common header file of CDC driver
               |          +——— src                       CDC driver
               +———SmplMain [ Sample Application ]
               |          +——— APL                       Sample 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+.

### 4.2.2 CDC File List

Table 4.2 shows the file structure supplied with PCDC.

**Table 4.2 PCDC Folders**

| Folder | File Name | Description | Note |
|---|---|---|---|
| PCDC/src | r_usb_pcdc_api.c | CDC API functions | |
| | r_usb_pcdc_driver.c | CDC driver functions | |
| PCDC/inc | r_usb_pcdc_define.h | CDC type definitions and macro definitions | |
| | r_usb_pcdc_extern.h | CDC prototype, external reference | |
| SmplMain | main.c | Main function | |
| SmplMain/APL | r_usb_pcdc_echo_apl.c | Sample application program for echo mode | |
| | r_usb_pcdc_uart_apl.c | Sample application program for Serial-USB converter mode | |
| | r_usb_pcdc_descriptor.c | PCDC descriptor for Sample application | |

## 4.3 System Resources

Table 4.3 lists the ID and priority definitions used to register PCDC in the scheduler.

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

**Table 4.3 Resource Definitions**

| | Name | Description |
|---|---|---|
| Scheduler registration task | USB_PCDC_TSK | **PCDC task** (usb_pcdc_Task)<br>Task priority: 1 |
| | USB_PCDCSMP_TSK | **APL main task** (usb_pcdc_main_task)<br>Task priority: 2 |
| | USB_PCD_TSK | **PCD task** (R_usb_pstd_PcdTask)<br>Task priority : 0 |
| Mailbox ID | USB_PCDC_MBX<br>(default value:<br>USB_PCDC_TSK) | PCDC mailbox ID |
| | USB_PCDCSMP_MBX<br>(default value:<br>USB_PCDCSMP_TSK) | APL mailbox ID |
| | USB_PCD_MBX<br>(default value:<br>USB_PCD_TSK) | **PCD** mailbox ID |

# 5. Peripheral CDC Sample Application (APL)

This section explains the peripheral CDC Sample Application (APL).

## 5.1 Operating Environment

Figure 5-1shows the sample operating environment for the software.

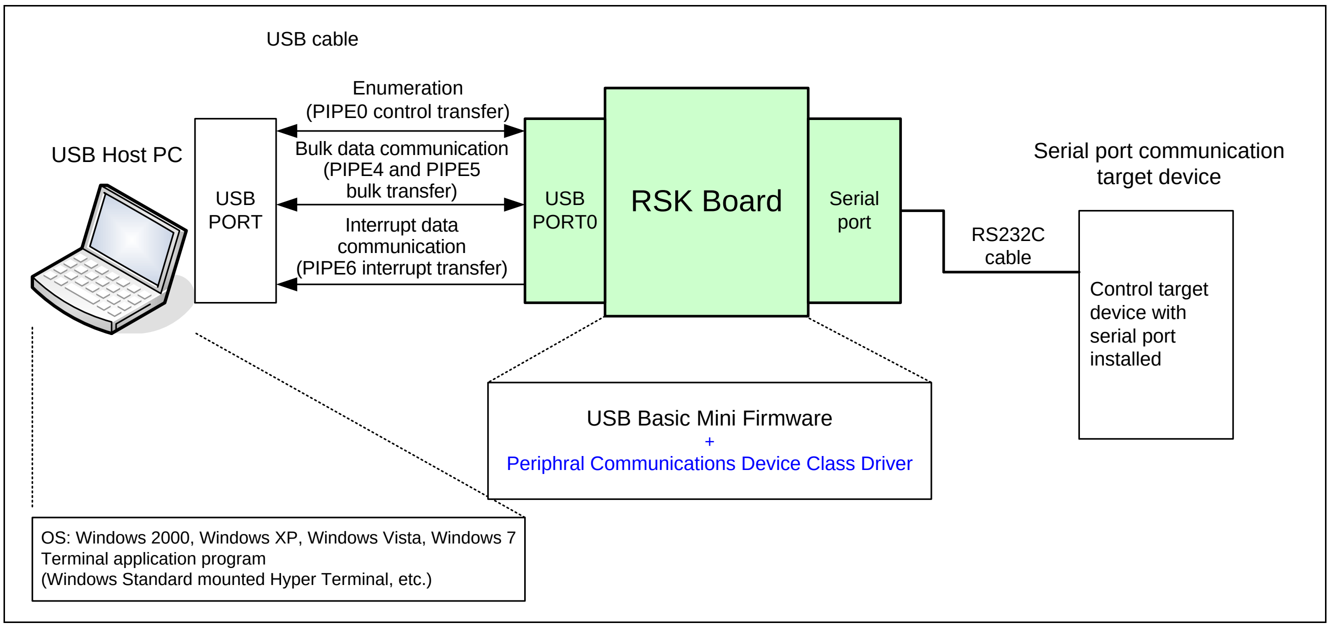

USB cable
Enumeration
(PIPE0 control transfer)
Bulk data communication
(PIPE4 and PIPE5
bulk transfer)
Interrupt data
communication
(PIPE6 interrupt transfer)
USB Host PC
USB PORT
USB PORT0
RSK Board
Serial port
Serial port communication target device
RS232C cable
Control target device with serial port installed
USB Basic Mini Firmware
+
Periphral Communications Device Class Driver
OS: Windows 2000, Windows XP, Windows Vista, Windows 7
Terminal application program
(Windows Standard mounted Hyper Terminal, etc.)

**Figure 5-1 Example Operating Environment**

When using a Windows PC as the Host PC, you will also need to install the system definition file (reference\cdc_inf\CDC_Demo.inf or CDC_Demo_Win7.inf). The system definition file must be edited to match the Vendor ID (VID) and Product ID (PID) setup in the Rev.2.15. Edit the following places in the system definition file using a text editor or similar editing tool.

[Model]

%STRING_MODEL% = CDC, USB\VID_0000&PID_0000 ← Edit the 4-digit numeric values as 4-digit hexadecimal numbers.

Edit the line as follows for a VID of 0x1234 and a PID of 0x5678.

[Model]

%STRING_MODEL% = CDC, USB\VID_1234&PID_5678 ← Edit the 4-digit numeric values as 4-digit hexadecimal numbers.

The PID and VID values of the peripheral device are defined by USB_VENDORID and USB_PRODUCTID, respectively, in the file WorkSpace\SmplMain\APL\r_usb_echo_apl_descriptor.c.

## 5.2 Application Program (APL) Overview

The application program works in the following 2 mode. The files of the application program is differ in each mode differ. Refer to chapter 5.2.3 about selecting the mode.

### 5.2.1 Serial-USB converter mode

The board works as a USB-to-serial (UART) converter. Incoming data from the USB host is sent to the UART port of the board. Conversely, incoming data to the board UART is sent to the USB COM-port, that is, the USB host.

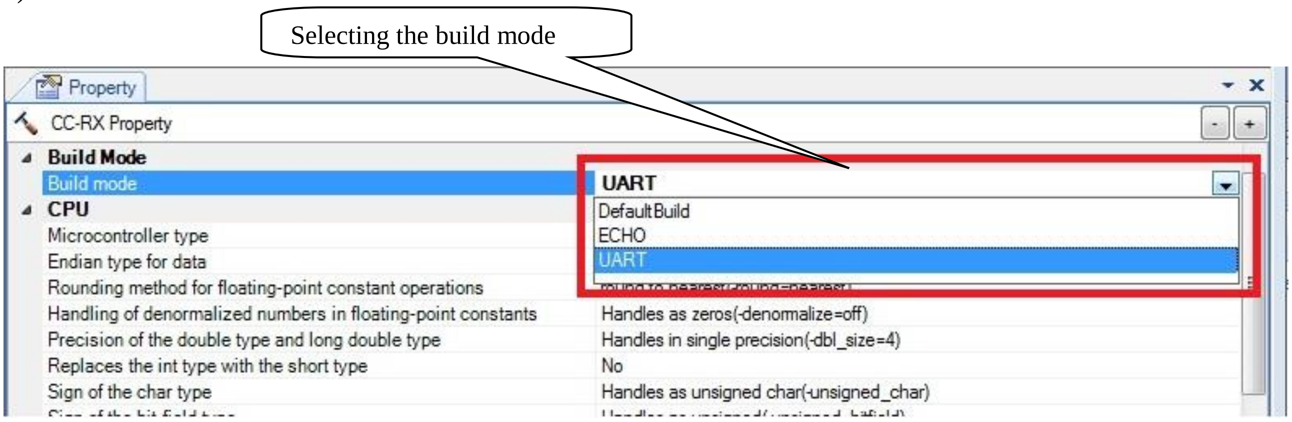
### 5.2.2 Echo Mode

Echo mode transmits by return the data received from the USB host to a USB host. A UART port is not used in echo mode.

### 5.2.3 Selecting Serial-USB converter mode / Echo mode

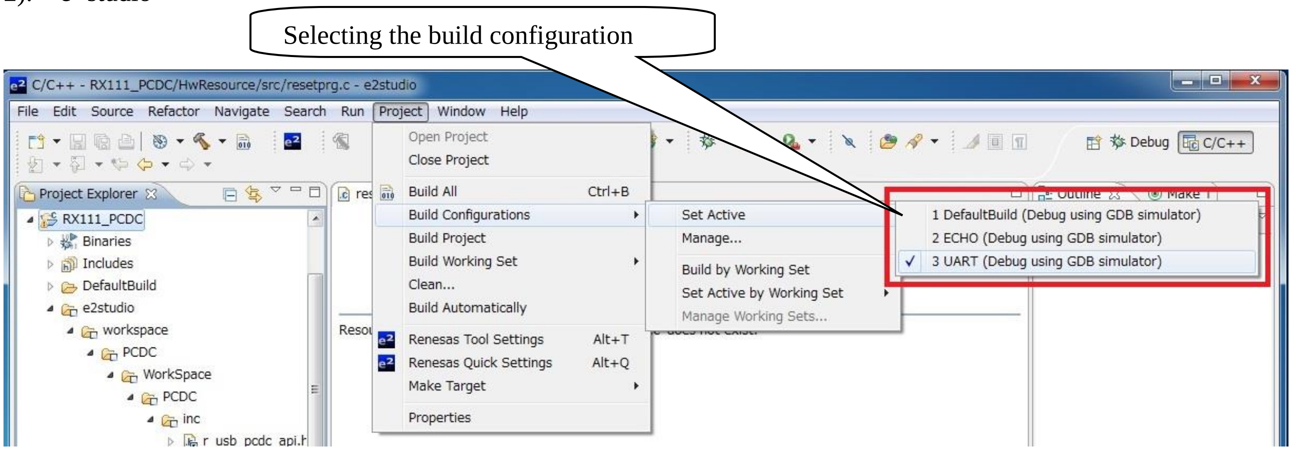
Select “Serial-USB converter mode / Echo mode” on the integrated development environment (IDE) after starting the IDE is supported by each MCU.

1). CS+



| | |
|---|---|
| **Build Mode** | |
| Build mode | UART |
| **CPU** | |
| Microcontroller type | |
| Endian type for data | |
| Rounding method for floating-point constant operations | |
| Handling of denormalized numbers in floating-point constants | Handles as zeros(-denormalize=off) |
| Precision of the double type and long double type | Handles in single precision(-dbl_size=4) |
| Replaces the int type with the short type | No |
| Sign of the char type | Handles as unsigned char(-unsigned_char) |

2). e$^2$ studio


Selecting the build configuration
C/C++ - RX111_PCDC/HwResource/src/resetprg.c - e2studio
File Edit Source Refactor Navigate Search Run Project Window Help
Project Explorer
RX111_PCDC
Binaries
Includes
DefaultBuild
e2studio
workspace
PCDC
WorkSpace
PCDC
inc
r_usb_pcdc_api.h
Open Project
Close Project
Build All Ctrl+B
Build Configurations
Build Project
Build Working Set
Clean...
Build Automatically
Renesas Tool Settings Alt+T
Renesas Quick Settings Alt+Q
Make Target
Properties
Set Active
Manage...
Build by Working Set
Set Active by Working Set
Manage Working Sets...
1 DefaultBuild (Debug using GDB simulator)
2 ECHO (Debug using GDB simulator)
✓ 3 UART (Debug using GDB simulator)
Debug
C/C++
Outline
Make

## 5.3 APL Messages

The application module (APL) receives messages from the mailbox USB_PCDCSMP_MBX). APL then processes the messages as described in “Table 5.1 APL Receive Message List”.

**Table 5.1 APL Receive Message List**

| Classification | Source of Message |
|---|---|
| USB_PCDC_RX_COMP | The callback function called at completion of USB reception (OUT): “usb_psmpl_RxCB” |
| USB_PCDC_TX_COMP | The callback function called at completion of a USB transmission (IN): “usb_psmpl_TxCB” |
| USB_PCDC_STATUS_TX_COMP | A serial state transmission completion: “usb_psmpl_state_notification” |
| USB_PCDC_PERIODIC | The cyclical start signal to process the sample application task: “usb_psmpl_periodic_request” |

## 5.4 APL Functions

Table 5.2 lists and describes the APL level functions.

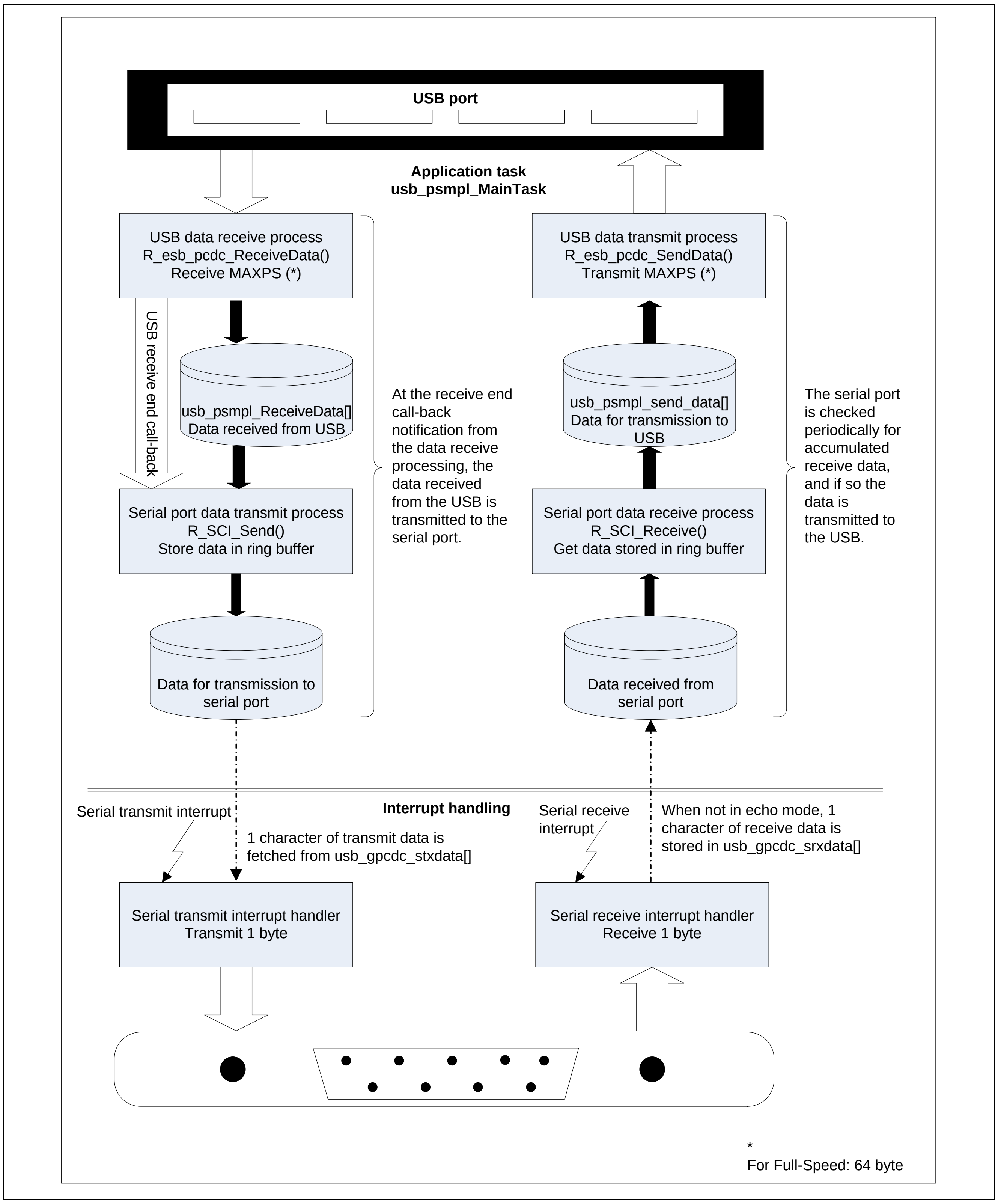
**Table 5.2 Lists of APL Functions**

| Function Name | Description |
|---|---|
| usb_cstd_task_start | Task start processing |
| usb_pcdc_task_start | Various task start process for peripheral USB |
| usb_psmpl_driver_registration | PCDC driver registration |
| usb_psmpl_open | PCDC open function |
| usb_psmpl_close | PCDC close function |
| usb_apl_task_switch | The task-switching loop |
| usb_psmpl_MainTask | Sample application main task |
| usb_psmpl_RxCB | The completion callback function of USB reception |
| usb_psmpl_TxCB | The completion callback function of USB transmitting |
| usb_psmpl_GetRcvDataCnt | Receive data count acquisition processing |
| usb_psmpl_change_device_state | Device state callback check |
| usb_psmpl_ReceiveDataStart | Start the date receive request for Host |
| usb_psmpl_LineCodingInitial | Line Coding initial processing |
| usb_psmpl_DummyFunc | Dummy function for the callback |
| usb_psmpl_state_notification | Callback function for notifying the serial state |
| usb_psmpl_class_request_callback | Callback function for receiving the class request |
| usb_psmpl_periodic_request | Application program cyclic start request |
| usb_psmpl_uart_callback | Callback function for UART driver |
| usb_psmpl_serial_state_process | Serial State processing |
| usb_psmpl_is_connected | Return the USB connection state |

## 5.5 APL Dataflow

An application level dataflow overview is shown in Figure 5-2.

### 5.5.1 Serial-USB converter mode



USB port
Application task
usb_psmpl_MainTask
USB data receive process
R_esb_pcdc_ReceiveData()
Receive MAXPS (*)
USB data transmit process
R_esb_pcdc_SendData()
Transmit MAXPS (*)
USB receive end call-back
usb_psmpl_ReceiveData[]
Data received from USB
usb_psmpl_send_data[]
Data for transmission to USB
At the receive end call-back notification from the data receive processing, the data received from the USB is transmitted to the serial port.
The serial port is checked periodically for accumulated receive data, and if so the data is transmitted to the USB.
Serial port data transmit process
R_SCI_Send()
Store data in ring buffer
Serial port data receive process
R_SCI_Receive()
Get data stored in ring buffer
Data for transmission to serial port
Data received from serial port
Interrupt handling
Serial transmit interrupt
1 character of transmit data is fetched from usb_gpcdc_stxdata[]
Serial receive interrupt
When not in echo mode, 1 character of receive data is stored in usb_gpcdc_srxdata[]
Serial transmit interrupt handler
Transmit 1 byte
Serial receive interrupt handler
Receive 1 byte
*
For Full-Speed: 64 byte

**Figure 5-2 APL Data Flow (Serial-USB converter mode)**

### 5.5.2 Echo Mode

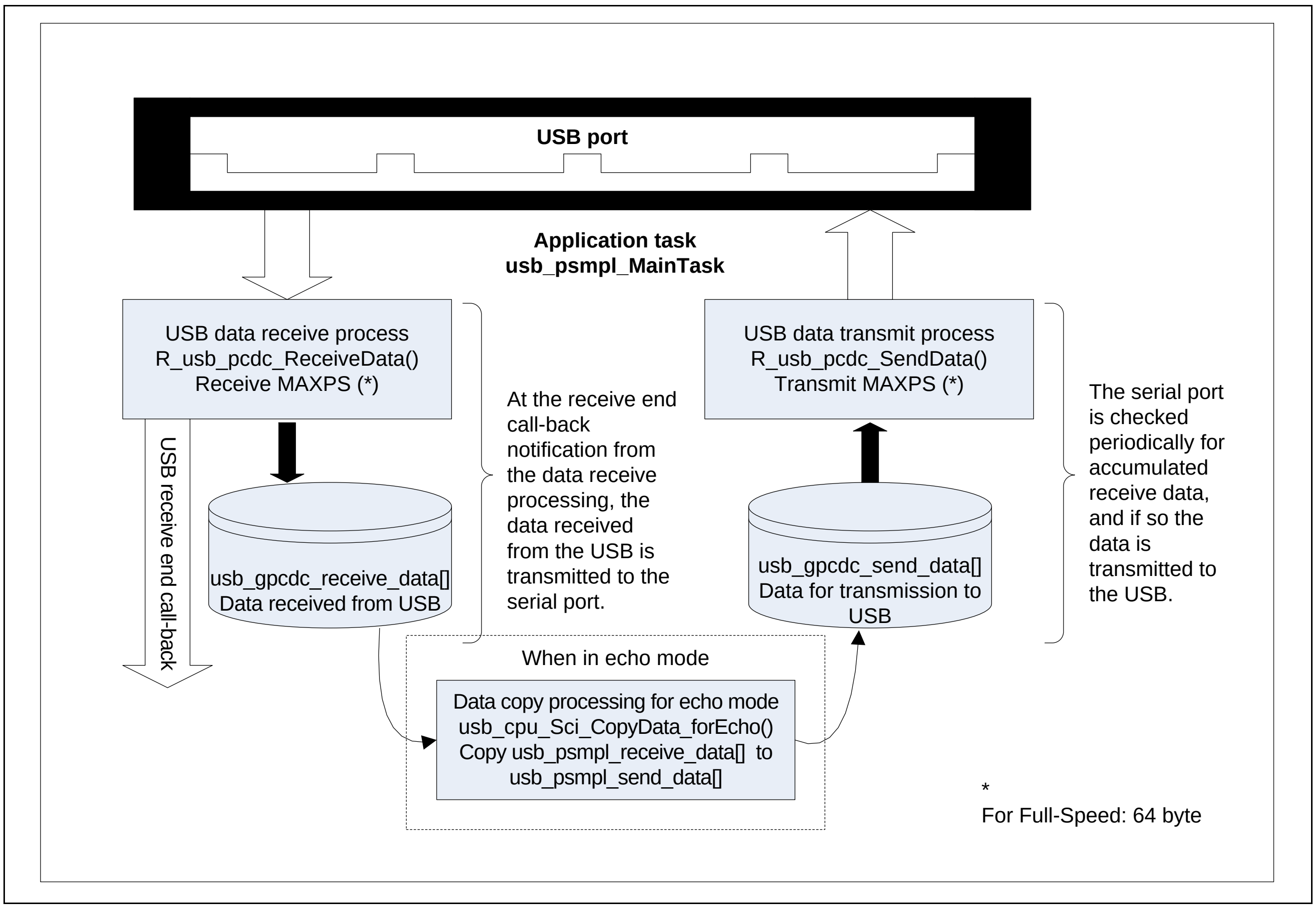

USB port
Application task
usb_psmpl_MainTask
USB data receive process
R_usb_pcdc_ReceiveData()
Receive MAXPS (*)
USB data transmit process
R_usb_pcdc_SendData()
Transmit MAXPS (*)
USB receive end call-back
usb_gpcdc_receive_data[]
Data received from USB
At the receive end call-back notification from the data receive processing, the data received from the USB is transmitted to the serial port.
usb_gpcdc_send_data[]
Data for transmission to USB
The serial port is checked periodically for accumulated receive data, and if so the data is transmitted to the USB.
When in echo mode
Data copy processing for echo mode
usb_cpu_Sci_CopyData_forEcho()
Copy usb_psmpl_receive_data[] to usb_psmpl_send_data[]
*
For Full-Speed: 64 byte

**Figure 5-3 APL Data Flow (Echo mode)**

## 5.6 Sequence Charts

Below are time sequence charts showing the interaction between the modules APL (application), PCDC (device class driver), PCD (USB device HW control) and CPD (serial port control driver).

### 5.6.1 Normal Mode (Serial-USB converter mode)

#### 1. Reception from CDC Host => Serial Port Transmit

The sequence whereby data is received from the CDC host and then transmitted to the serial port is shown Figure 5-4.

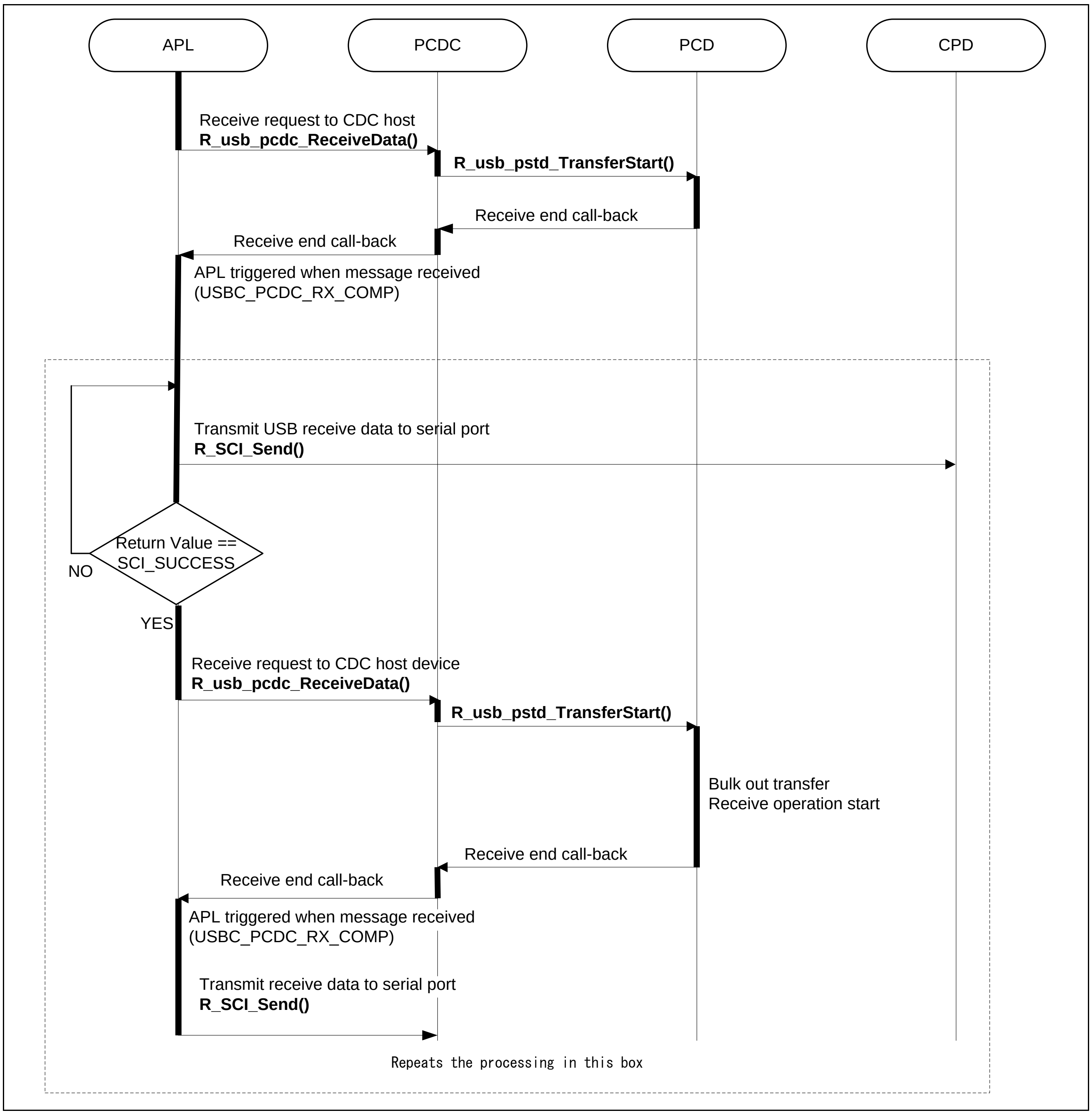

APL
PCDC
PCD
CPD
Receive request to CDC host
R_usb_pcdc_ReceiveData()
R_usb_pstd_TransferStart()
Receive end call-back
Receive end call-back
APL triggered when message received
(USBC_PCDC_RX_COMP)
Transmit USB receive data to serial port
R_SCI_Send()
Return Value == SCI_SUCCESS
NO
YES
Receive request to CDC host device
R_usb_pcdc_ReceiveData()
R_usb_pstd_TransferStart()
Bulk out transfer
Receive operation start
Receive end call-back
Receive end call-back
APL triggered when message received
(USBC_PCDC_RX_COMP)
Transmit receive data to serial port
R_SCI_Send()
Repeats the processing in this box

**Figure 5-4 Data Reception from CDC Host and Serial Port Transmit Sequence**

## 2. Serial Port Reception => Transmission To CDC Host

The sequence whereby is data received from the serial port and then transmitted to the USB Host is shown Figure 5-8.

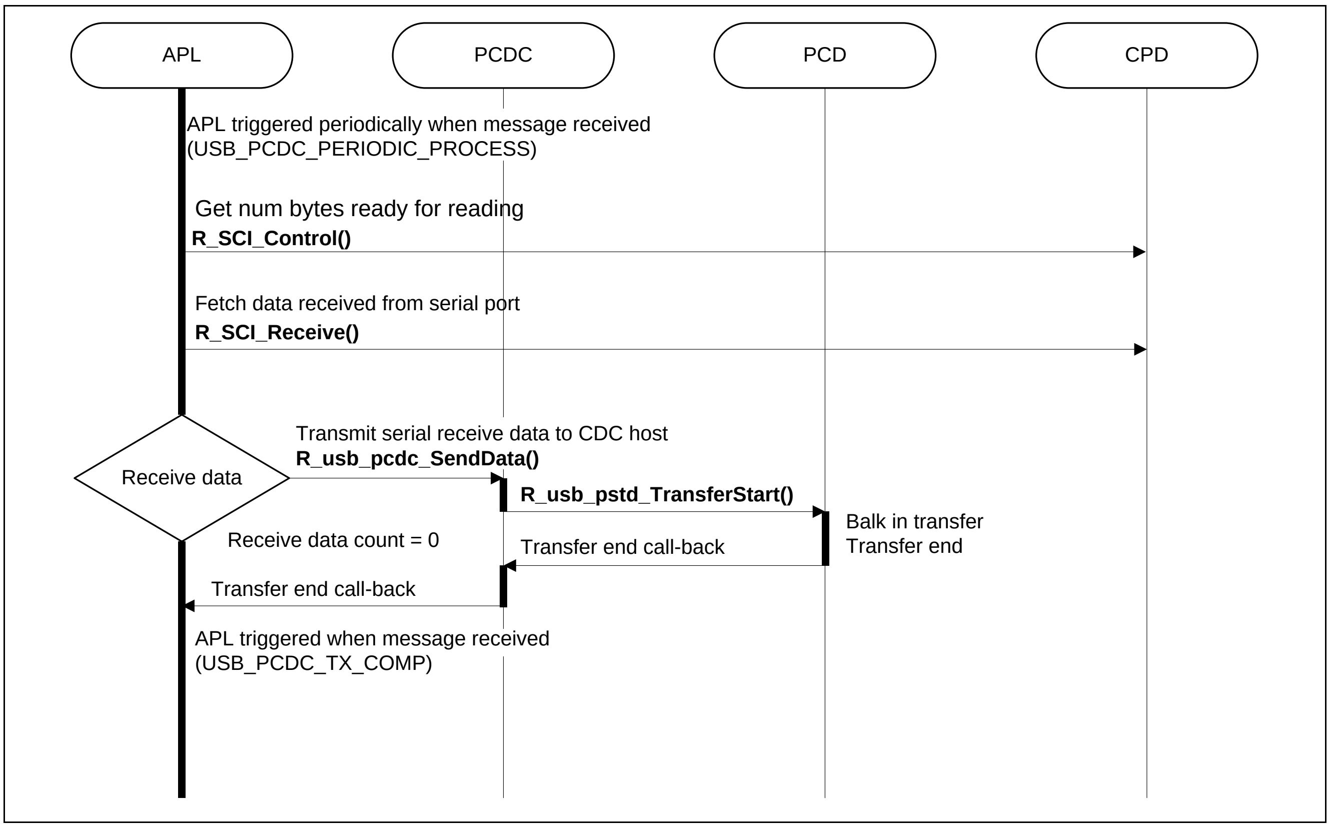

APL
PCDC
PCD
CPD
APL triggered periodically when message received (USB_PCDC_PERIODIC_PROCESS)
Get num bytes ready for reading
R_SCI_Control()
Fetch data received from serial port
R_SCI_Receive()
Receive data
Transmit serial receive data to CDC host
R_usb_pcdc_SendData()
R_usb_pstd_TransferStart()
Balk in transfer
Transfer end
Receive data count = 0
Transfer end call-back
Transfer end call-back
APL triggered when message received (USB_PCDC_TX_COMP)

**Figure 5-5 Serial Port (UART) Reception and Transmission to CDC Host Sequence**

## 3. Serial Error Handling

The sequence when a serial receive error is detected, and a class notification (SerialState) is transmitted to the USB Host, is shown Figure 5-6.

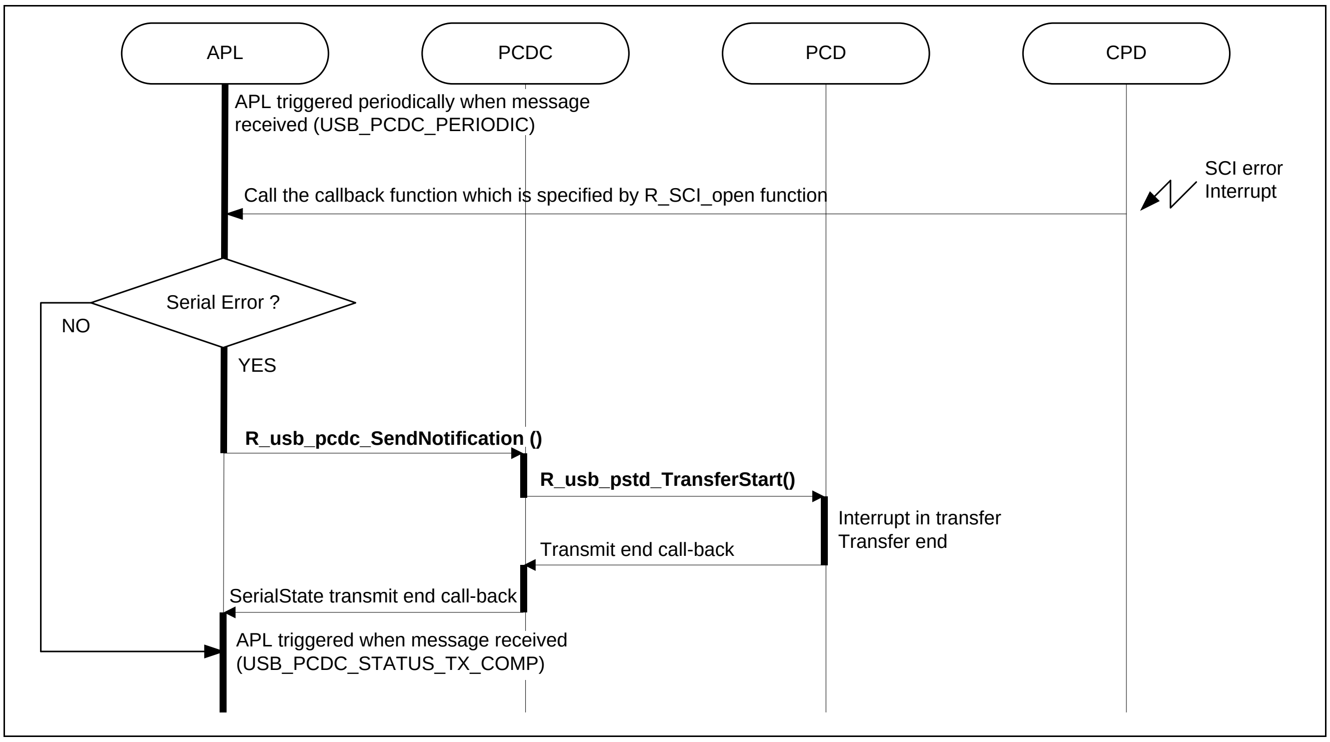

APL
PCDC
PCD
CPD
APL triggered periodically when message received (USB_PCDC_PERIODIC)
Call the callback function which is specified by R_SCI_open function
SCI error Interrupt
Serial Error ?
NO
YES
R_usb_pcdc_SendNotification ()
R_usb_pstd_TransferStart()
Interrupt in transfer Transfer end
Transmit end call-back
SerialState transmit end call-back
APL triggered when message received (USB_PCDC_STATUS_TX_COMP)

**Figure 5-6 Serial Error Handling Sequence**

### 5.6.2 Echo Mode

The sequence of echo mode operation, in which data received from the USB Host is transmitted back to the USB Host, is shown Figure 5-7.

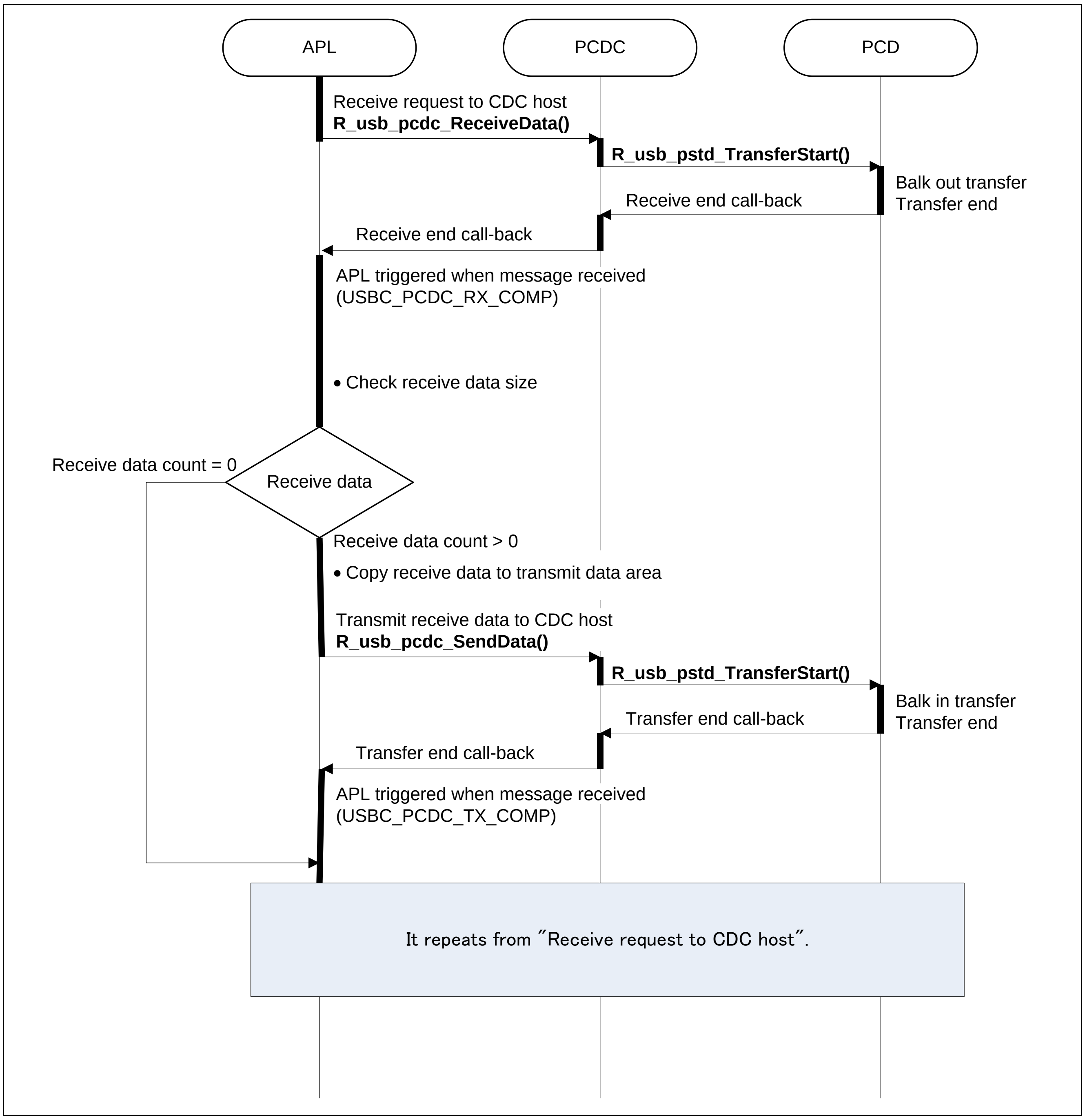

APL
PCDC
PCD
Receive request to CDC host
R_usb_pcdc_ReceiveData()
R_usb_pstd_TransferStart()
Balk out transfer
Transfer end
Receive end call-back
Receive end call-back
APL triggered when message received
(USBC_PCDC_RX_COMP)
• Check receive data size
Receive data count = 0
Receive data
Receive data count > 0
• Copy receive data to transmit data area
Transmit receive data to CDC host
R_usb_pcdc_SendData()
R_usb_pstd_TransferStart()
Balk in transfer
Transfer end
Transfer end call-back
Transfer end call-back
APL triggered when message received
(USBC_PCDC_TX_COMP)
It repeats from "Receive request to CDC host".

**Figure 5-7 Echo Mode Sequence**

## 5.7 APL Processing Details

Here follows a more detailed description of the processing pathways for USB-to-serial-UART mode and Echo mode.

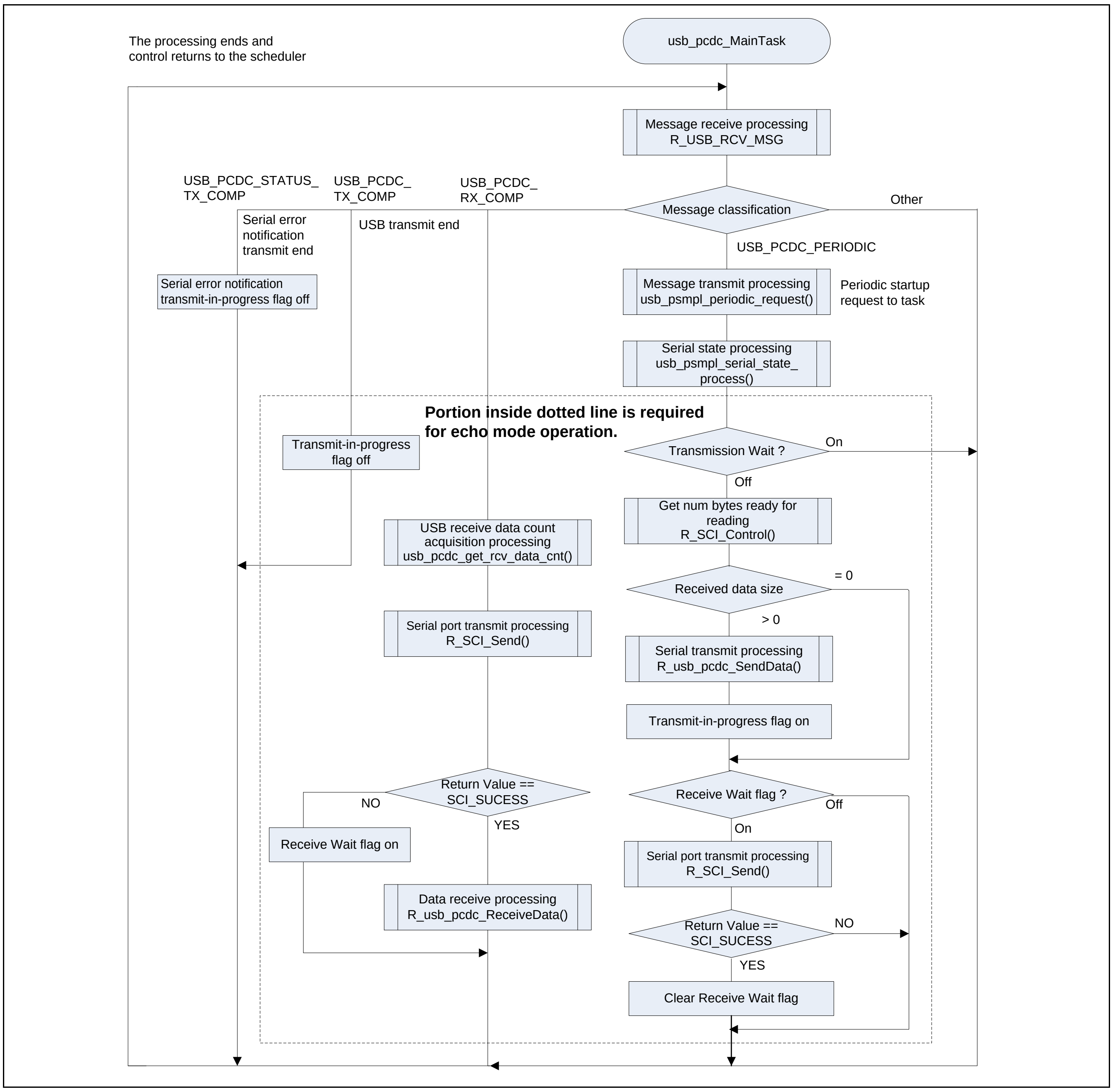
### 1. USB – Serial converter processing

・ USB → Serial (UART)

①. USB data reception processing done by: *R_usb_pcdc_ReceiveData()*
②. Transmission over serial port: *R_SCI_Send()*
①and ② repeated.

・ Serial (UART) → USB

③. Serial receive processing done by: *R_usb_sci_receive()*
④. USB transmission by: *R_usb_sci_send()*
③ and ④ repeated.

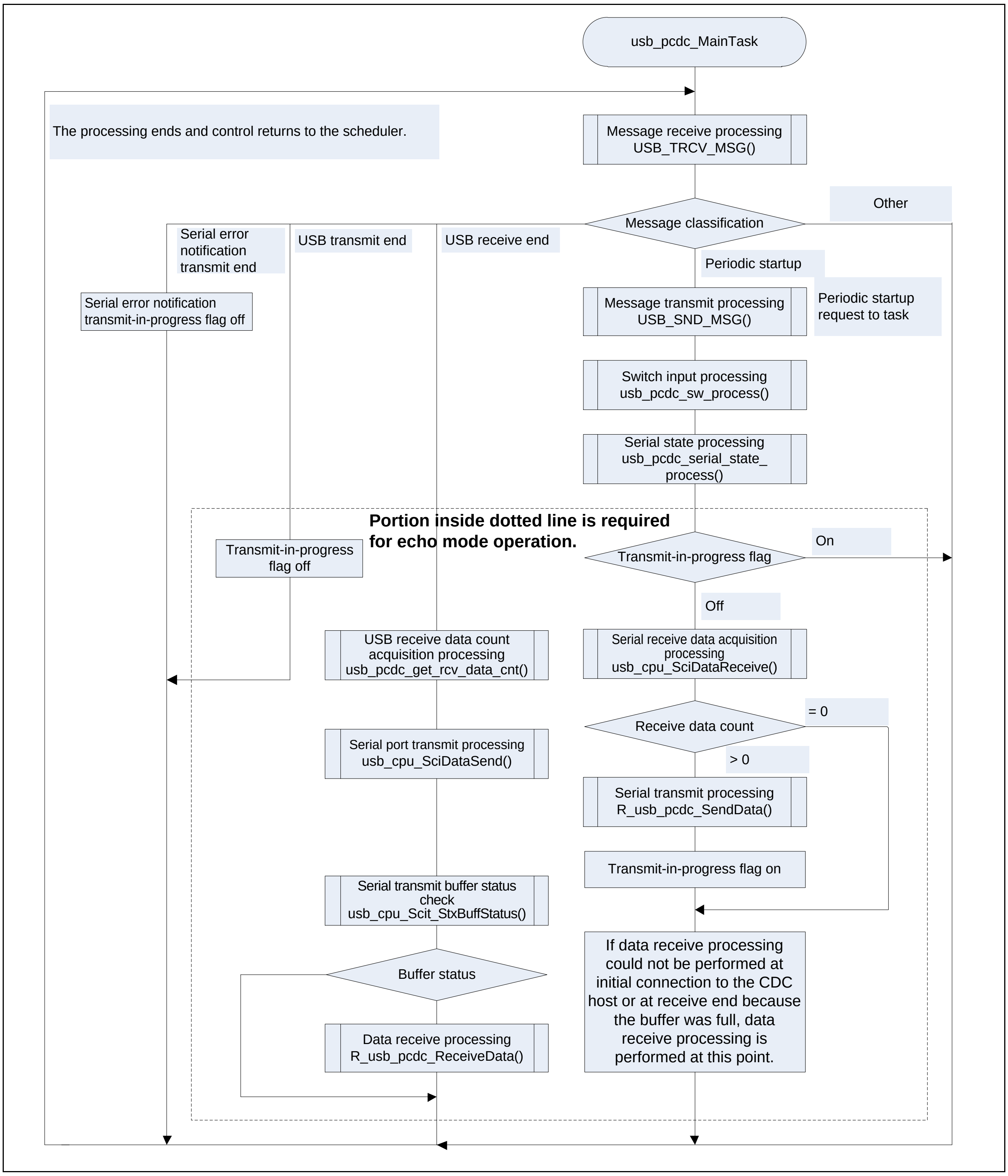


The processing ends and control returns to the scheduler
usb_pcdc_MainTask
Message receive processing R_USB_RCV_MSG
USB_PCDC_STATUS_TX_COMP
USB_PCDC_TX_COMP
USB_PCDC_RX_COMP
Message classification
Other
Serial error notification transmit end
USB transmit end
USB_PCDC_PERIODIC
Serial error notification transmit-in-progress flag off
Message transmit processing usb_psmpl_periodic_request()
Periodic startup request to task
Serial state processing usb_psmpl_serial_state_process()
Portion inside dotted line is required for echo mode operation.
Transmit-in-progress flag off
Transmission Wait ?
On
Off
Get num bytes ready for reading R_SCI_Control()
USB receive data count acquisition processing usb_pcdc_get_rcv_data_cnt()
Received data size
= 0
> 0
Serial port transmit processing R_SCI_Send()
Serial transmit processing R_usb_pcdc_SendData()
Transmit-in-progress flag on
Return Value == SCI_SUCESS
NO
YES
Receive Wait flag on
Receive Wait flag ?
Off
On
Serial port transmit processing R_SCI_Send()
Data receive processing R_usb_pcdc_ReceiveData()
Return Value == SCI_SUCESS
NO
YES
Clear Receive Wait flag

**Figure 5-8 APL Processing Flowchart (USB – Serial converter Mode)**

## 2. Echo Mode

① Received USB data processing: *usb_psmpl_ReceiveDataStart()*
② USB transmission: *R_usb_pcdc_SendData()*
①and ② repeated.



usb_pcdc_MainTask
The processing ends and control returns to the scheduler.
Message receive processing USB_TRCV_MSG()
Other
Message classification
Serial error notification transmit end
USB transmit end
USB receive end
Periodic startup
Serial error notification transmit-in-progress flag off
Message transmit processing USB_SND_MSG()
Periodic startup request to task
Switch input processing usb_pcdc_sw_process()
Serial state processing usb_pcdc_serial_state_process()
Portion inside dotted line is required for echo mode operation.
Transmit-in-progress flag off
Transmit-in-progress flag
On
Off
USB receive data count acquisition processing usb_pcdc_get_rcv_data_cnt()
Serial receive data acquisition processing usb_cpu_SciDataReceive()
Receive data count
= 0
> 0
Serial port transmit processing usb_cpu_SciDataSend()
Serial transmit processing R_usb_pcdc_SendData()
Transmit-in-progress flag on
Serial transmit buffer status check usb_cpu_Scit_StxBuffStatus()
Buffer status
Data receive processing R_usb_pcdc_ReceiveData()
If data receive processing could not be performed at initial connection to the CDC host or at receive end because the buffer was full, data receive processing is performed at this point.

**Figure 5-9 APL Processing Flowchart (Echo Mode)**

# 6. Communications Device Class (CDC)

## 6.1 Basic Functions

This software conforms to the Abstract Control of the CDC PSTN Subclass.

The main functions of this "PCDC" firmware are as follows.

1. Respond to functional inquiries from the USB Host
2. Respond to class requests from the USB Host
3. Data communication with the USB Host
4. Notifying the USB Host of serial communication errors

## 6.2 Abstract Control Model Overview

The Abstract Control Model subclass of CDC 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. Nowadays, the ACM subclass is used for USB applications that need USB bulk transfer – for larger amounts of non time critical data. The ACM is ideal for example for applications where a file needs to be transferred. This can then be done using a PC UART terminal program that has a built-in file transfer menu option.

The class requests and class notifications supported are listed below.

### 6.2.1 Class Requests (Host to Peripheral)

Table 6.1 shows CDC class requests, and whether they are supported.

**Table 6.1 CDC class requests**

| Request | Code | Description | Supported |
|---|---|---|---|
| SendEncapsulatedCommand | 0x00 | Transmits AT commands, etc., defined by the protocol. | No* |
| GetEncapsulatedResponse | 0x01 | Requests a response to a command transmitted by SendEncapsulatedCommand. | No* |
| SetCommFeature | 0x02 | Enables or disables features such as device-specific 2-byte code and country setting. | No* |
| GetCommFeature | 0x03 | Acquires the enabled/disabled state of features such as device-specific 2-byte code and country setting. | No* |
| ClearCommFeature | 0x04 | Restores the default enabled/disabled settings of features such as device-specific 2-byte code and country setting. | No* |
| 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. | No* |

*Must be added by the user of PCDC.(Refer to the sample function in )

For details concerning 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.

### 6.2.2 Data Format of Class Requests

The data formats of the class requests supported by the class driver software are described below.

**(1). SetLineCoding**

This is the class request that the host transmits for UART line setting.

The SetLineCoding data format is shown below.

**Table 6.2 SetLineCoding Format**

| bmRequestType | bRequest | wValue | wIndex | wLength | Data |
|---|---|---|---|---|---|
| 0x21 | SET_LINE_CODING (0x20) | 0x00 | 0x00 | 0x07 | Line Coding Structure<br>See Table 6.3 Line Coding Format |

**Table 6.3 Line Coding Format**

| Offset | Field | Size | Value | Description |
|---|---|---|---|---|
| 0 | DwDTERate | 4 | Number | Data terminal speed (bps) |
| 4 | BcharFormat | 1 | Number | Stop bits 0 - 1 stop bit<br>1 - 1.5 stop bits<br>2 - 2 stop bits |
| 5 | BparityType | 1 | Number | Parity 0 - None<br>1 - Odd<br>2 - Even |
| 6 | BdataBits | 1 | Number | Data bits (5, 6, 7, 8) |

The following shows the setting that this S/W supports.

| | |
|---|---|
| DwDTERate: | 1200bps/2400bps/4800bps/9600bps/14400bps/19200bps/38400bps/57600bps/115200bps |
| BcharFormat: | 1 Stop bit/ 2 Stop bit |
| BparityType: | None/Odd/Even |
| BdataBits: | 7bit/8bit |

**(2). GetLineCoding**

This is the class request the host transmits to request the UART line state.

The GetLineCoding data format is shown below.

**Table 6.4 SetLineCoding Format**

| bmRequestType | bRequest | wValue | WIndex | wLength | Data |
|---|---|---|---|---|---|
| 0xA1 | GET_LINE_CODING (0x21) | 0x00 | 0x00 | 0x07 | Line Coding Structure<br>See table 4.3, Line Coding Structure Format |

### (3). SetControlLineSTATE

This is a class request that the host sends to set up the signal for flow controls of UART.

This software does not support RTS/DTR control.

The SET_CONTROL_LINE_STATE data format is shown below.

**Table 6.5 SET_CONTROL_LINE_STATE Format**

| bmRequestType | bRequest | wValue | wIndex | wLength | Data |
|---|---|---|---|---|---|
| 0x21 | SET_CONTROL_ LINE_STATE (0x22) | Control Signal Bitmap See Table 4.6, Control Signal Bitmap Format | 0x00 | 0x00 | None |

**Table 6.6 Control Signal Bitmap**

| Bit Position | Description | |
|---|---|---|
| D15 to D2 | Reserved (reset to 0) | |
| D1 | DCE transmit function control | 0 - RTS Off<br>1 - RTS On |
| D0 | Notification of DTE ready state | 0 - DTR Off<br>1 - DTR On |

### 6.2.3 Class Notifications (Peripheral to Host)

Whether or not a class notification is supported is shown in Table 6.7.

**Table 6.7 CDC Class Notifications**

| Notification | Code | Description | Supported |
|---|---|---|---|
| NETWORK_CONNECTION | 0x00 | Notification of network connection state | No |
| RESPONSE_AVAILABLE | 0x01 | Response to GET_ENCAPSLATED_RESPONSE | No |
| SERIAL_STATE | 0x20 | Notification of serial line state | Yes |

**(1). Serial State**

The host is notified of the serial state when a change in the UART port state is detected.

This software supports the detection of overrun, parity and framing errors. A state notification is performed when a change from normal state to error is detected. However, notification is not continually transmitted when an error is continually detected.

The SerialState data format is shown below.

**Table 6.8 SerialState Format**

| bmRequestType | bRequest | wValue | wIndex | wLength | Data |
|---|---|---|---|---|---|
| 0xA1 | SERIAL_STATE (0x20) | 0x00 | 0x00 | 0x02 | UART State bitmap See Table 6.9 UART state bitmap format |

**Table 6.9 UART state bitmap format**

| Bits | Field | Description | Supported |
|---|---|---|---|
| D15～D7 | | Reserved | - |
| D6 | bOverRun | Overrun error detected | Yes |
| D5 | bParity | Parity error detected | Yes |
| D4 | bFraming | Framing error detected | Yes |
| D3 | bRingSignal | INCOMING signal (ring signal) detected | No |
| D2 | bBreak | Break signal detected | No |
| D1 | bTxCarrier | Data Set Ready: Line connected and ready for communication | No |
| D0 | bRxCarrier | Data Carrier Detect: Carrier detected on line | No |

## 6.3 Endpoint Specification

Endpoints used are shown in Table 6.10.

**Table 6.10 Endpoints**

| bEndpointAddress | | bmAttributes | wMaxPacketSize | Description |
|---|---|---|---|---|
| EP No | Direction | Transfer Method | Max Packet Size | |
| EP0 | In/Out | Control | 64 | Standard request, class request |
| EP1 | In | Bulk | 64 (Full Speed) | Data transfer from device to host |
| EP2 | Out | Bulk | 64 (Full Speed) | Data transfer from host to device |
| EP3 | In | Interrupt | 16 | State notification from device to host |

## 6.4 Usage as PC virtual COM port (reference)

The CDC device can be used as a virtual COM port when operating in Windows OS.

When using a PC installed with Windows OS, when the RSK board is connected, and after enumeration , the CDC class requests GetLineCoding and SetControlLineState from the USB host are processed, and then the CDC device is registered in the Windows Device Manager as a virtual COM port device.

Registering the CDC device as a virtual COM port in the Windows device manager enables data communication with the CDC device via a terminal app, such as Hyper Terminal, which comes standard with Windows XP.

When selecting the COM port number and setting the serial port options in the PC terminal applicaton, the UART settings are propagated to the firmware via the class request SetLineCoding.

Data input (or file transmission) from the terminal app window is transmitted to the RSK board using EP2; data from the RSK board side is transmitted to the PC using EP1.

The maximum packet size for Full-Speed is 64 bytes. Note that when the last packet of data received is the maximum packet size, and the terminal determines that there is continuous data, the received data may not be displayed in the terminal. If the received data is smaller than the maximum packet size, the data received up to that point is displayed in the terminal.

# 7. USB Peripheral Communication Device Class Driver (PCDC)

## 7.1 Basic Functions

The basic functions of PCDC are as follows.

1. Provides data transmission and reception services to the USB host.
2. Responds to CDC class requests.
3. Provides a CDC notification transmission service.

## 7.2 PCDC API Functions

Table 7.1 below show all the PCDC API functions.

**Table 7.1 API Functions**

| Function Name | Description |
|---|---|
| R_usb_pcdc_LineCodingInitial | Line Coding initialization |
| R_usb_pcdc_SendData | Sends a data transmit request message to the PCDC task. |
| R_usb_pcdc_ReceiveData | Sends a data receive request message to the PCDC task. |
| R_usb_pcdc_ClassRequest | Control transfer processing for CDC |
| R_usb_pcdc_task | The PCDC task |

## R_usb_pcdc_LineCodingInitial

### Transfer USB data

**Format**

|  |  |
|---|---|
| usb_er_t | R_usb_pcdc_LineCodingInitial ( usb_pcdc_LineCoding_t *linecoding ) |

**Argument**

|  |  |
|---|---|
| *linecoding | LineCoding setting data address |

**Return Value**

|  |  |
|---|---|
| USB_E_OK | Success |

**Description**

This function initializes a LineCoding.

**Note**

—

## R_usb_pcdc_SendData

### Transfer USB data

#### Format

```
void          R_usb_pcdc_SendData (  uint8_t*       table,
                                     usb_leng_t     size,
                                     usb_cbinfo_t   complete)
```

#### Argument

| | |
|---|---|
| *Table | Pointer to buffer containing data to transmit |
| size | Transfer size |
| complete | Process completion callback function |

#### Return Value

| | |
|---|---|
| — | — |

#### Description

This function transfers the specified USB data of the specified size from the address specified in the Transmit Data Address Table(1st argument).

When the transmission is done, the call-back function 'complete' is called.

#### Note

1. The USB transmit process results are found via the *usb_utr_t* pointer in the call-back function's arguments.
2. See "USB Communication Structure" (usb_utr_t) in the USB Basic Mini Firmware application note.

#### Example

```
void usb_apl_task( void )
{
  uint8_t    send_data[] = {0x01,0x02,0x03,0x04,0x05};  /* USB send data */
  uint16_t   size = 5;                                  /* Data size */

  R_usb_pcdc_SendData((uint8_t *)send_data, size, (usb_cbinfo_t)&usb_complete)
}

/* Callback function */
void  usb_complete( usb_utr_t *mess );
{
  /* Processing at the time of the completion of USB transmitting */
}
```

## R_usb_pcdc_ReceiveData

### Issue a data receive request to the USB driver (PCD)

**Format**

| | |
|---|---|
| void | R_usb_pcdc_ReceiveData (uint8_t *Table, usb_leng_t size, usb_cbinfo_t complete) |

**Argument**

| | |
|---|---|
| *Table | Pointer to transmmit data buffer address |
| size | Transfer size |
| complete | Process completion callback function |

**Return Value**

| | |
|---|---|
| — | — |

**Description**

This function requests a USB data transfer reception of the USB driver. One transfer is handled, after that a new request must be issued.

When the data of the size specified by 3rd argument is received or the data of less than max packet size is received from USB, callback function is called.

The received data is stored in the area that is specified by the second argument .

**Note**

1. The USB transfer results are found via the usb_utr_t pointer in the call-back function's arguments.
2. See "USB Communication Structure" (usb_utr_t) in the USB Basic Mini Firmwar application note.

**Example**

```
void usb_smp_task( void )
{
  uint8_t    receive_data[64];                      /* Data buff */
  uint16_t   size = 64;                             /* Data size */

  R_usb_pcdc_ReceiveData((uint8_t *)receive_data,size,
(usb_cbinfo_t)&usb_complete)
}

/* Callback function */
void  usb_complete( usb_utr_t *mess );
{
  /* Processing at the time of the completion of USB reception */
}
```

## R_usb_pcdc_ClassRequest

### Control transfer processing for CDC

**Format**

| | |
|---|---|
| void | R_usb_pcdc_ClassRequest(usb_request_t *request, uint16_t data) |

**Argument**

| | | |
|---|---|---|
| *request | Class request message pointer. | |
| data | Control transfer stage information | |
| | USB_CS_IDST | Idle or setup stage |
| | USB_CS_RDDS | Control read data stage |
| | USB_CS_WRDS | Control write data stage |
| | USB_CS_WRND | Control write no data status stage |
| | USB_CS_RDSS | Control read status stage |
| | USB_CS_WRSS | Control write status stage |
| | USB_CS_SQER | Sequence error |

**Return Value**

| | |
|---|---|
| — | — |

**Description**

When the request type is a CDC class request, this function calls the processing that corresponds to the control transmit stage.

This callback is registered earlier during PCDC class “driver registration”and is triggered at the end of a CDC control transfer.

**Note**

| | |
|---|---|
| — | — |

**Example**

```
void usb_apl_task( void )
{
  usb_pcdreg_t  driver;

        :
  /* Control Transfer */
  driver.ctrltrans = &R_usb_pcdc_ClassRequest;
  R_usb_pcdc_Registration(&driver);
        :
}
```

## R_usb_pcdc_Task

### The PCDC task

#### Format

|  |  |
|---|---|
| void | R_usb_pcdc_Task(void) |

#### Argument

|  |  |
|---|---|
| — | — |

#### Return Value

|  |  |
|---|---|
| — | — |

#### Description

This is the PCDC task, which processes requests by the application and then notifies it of  the results.

#### Note

In non-OS operations, the function is registered to be scheduled by the scheduler.

Refer to the USB-BASIC-F/WApplication Notes for more information concerning the scheduling process.

#### Example

```
void usb_apl_task_switch(void)
{
  while( 1 )
  {
    if( USB_FLGSET == R_usb_cstd_Schedule() )
    {
        /* PCD Task */
        R_usb_pstd_PcdTask();

        /* Peripheral Communications Devices Class Task */
        R_usb_pcdc_Task();

        /* Peripheral Communications Class Application Task */
        usb_pcdc_main_task();
    }
  }
}
```

## 7.3 User Definition Tables

It is necessary to create a descriptor table and Pipe Information Table (or “Endpoint Table”) for use by PCDC. Refer to the sample in file *r_usb_PCDCdescriptor.c* when creating these tables.

For details, see *Renesas USB Device USB Basic Mini Firmware User’s Manual*.

# 8. Communication Port Driver (CPD)

The communications port driver (CPD) is a serial communications driver for UART in the RSK.

When using this application on hardware other than the RSK, please prepare a matching serial communications driver.

## 8.1 RL78 Series

The serial communication driver which is used in RL78 series conform to the driver of the above application note (Document No. R01AN1667EU) which is used in RX series.

Refer to this application note about the API and the argument for the serial communication driver.

### 8.1.1 Overview of Functions

#### 1. Serial communication specification

(1) Line speed (1200 bps to 115200 bps)
(2) Parity bit (none, even, odd)
(3) Stop bits (1 or 2 bits)
(4) Data length (7 or 8 bits)

#### 2. Transmit

Transmitting data to the serial port involves saving the transmit data to the ring buffer of CPD and then transmitting the data one byte at a time, using the transmit data empty interrupt.

#### 3. Receive

Receiving data from the serial port involves saving data received using the receive data full interrupt to the ring buffer of CPD. The receive data is processed by using the receive data read API provided by CPD.

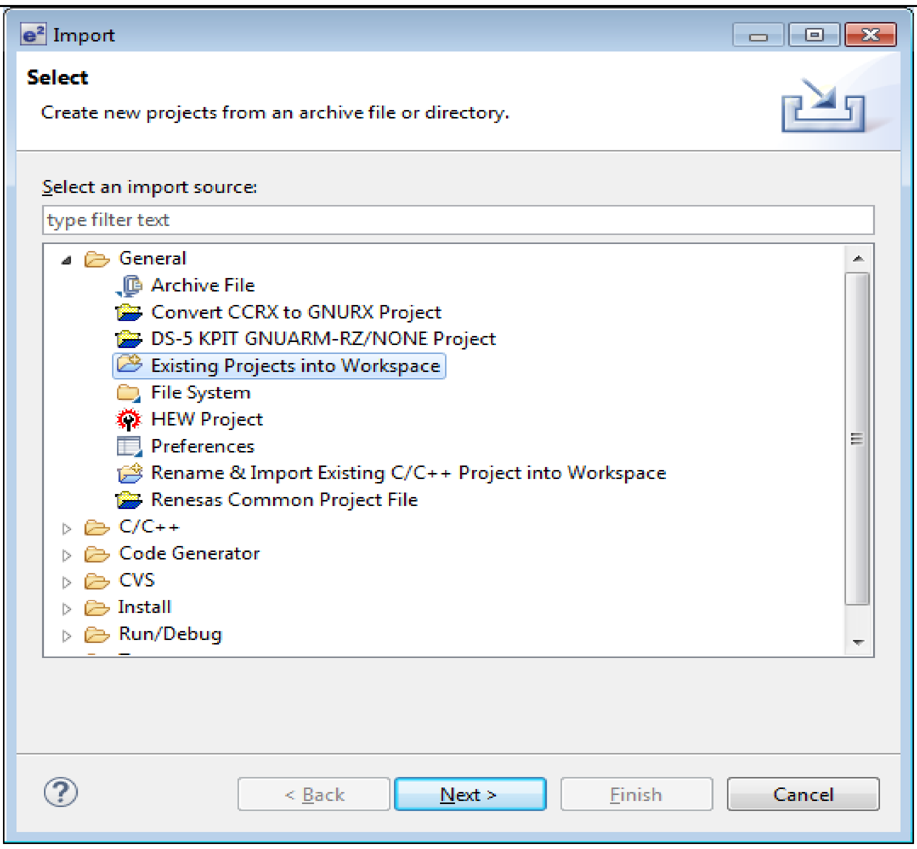
## 9. Setup for the $e^2$ studio project

(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].



Import

**Select**

Create new projects from an archive file or directory.

Select an import source:

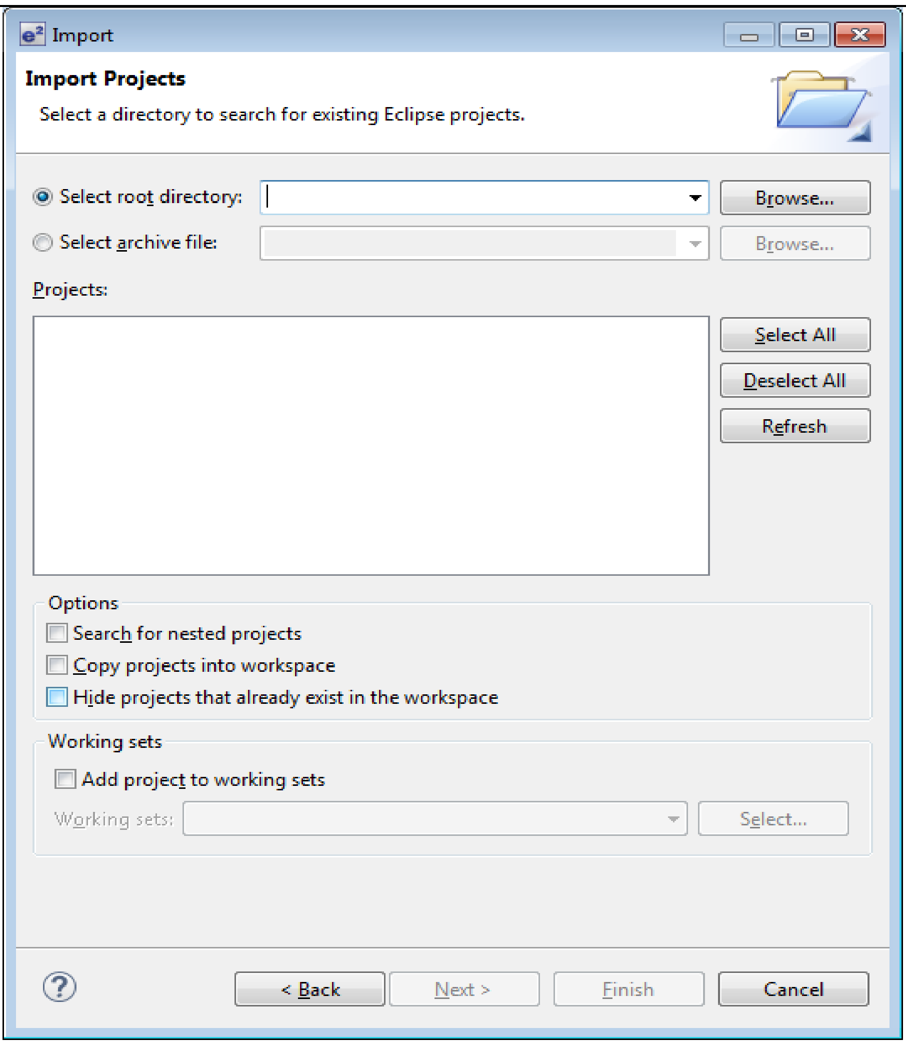
type filter text

- General
  - Archive File
  - Convert CCRX to GNURX Project
  - DS-5 KPIT GNUARM-RZ/NONE Project
  - Existing Projects into Workspace
  - File System
  - HEW Project
  - Preferences
  - Rename & Import Existing C/C++ Project into Workspace
  - Renesas Common Project File
- C/C++
- Code Generator
- CVS
- Install
- Run/Debug

< Back | Next > | Finish | Cancel

Figure 9-1 Select Import Source

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


Import

Import Projects

Select a directory to search for existing Eclipse projects.

Select root directory: [✓]

Select archive file:

Projects:

Options

- [ ] Search for nested projects
- [ ] Copy projects into workspace
- [ ] Hide projects that already exist in the workspace

Working sets

- [ ] Add project to working sets

Working sets:

**Figure 9-2 Project Import Dialog Box**

(5). Click [Finish].

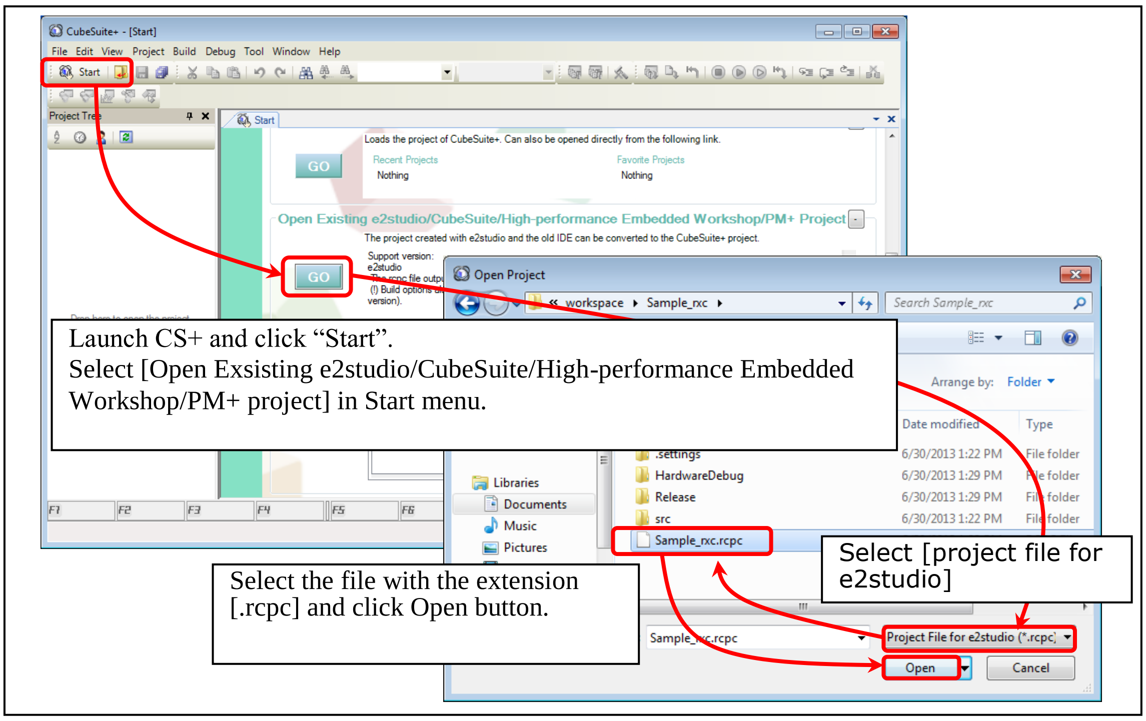
This completes the step for importing a project to the project workspace.

## 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.

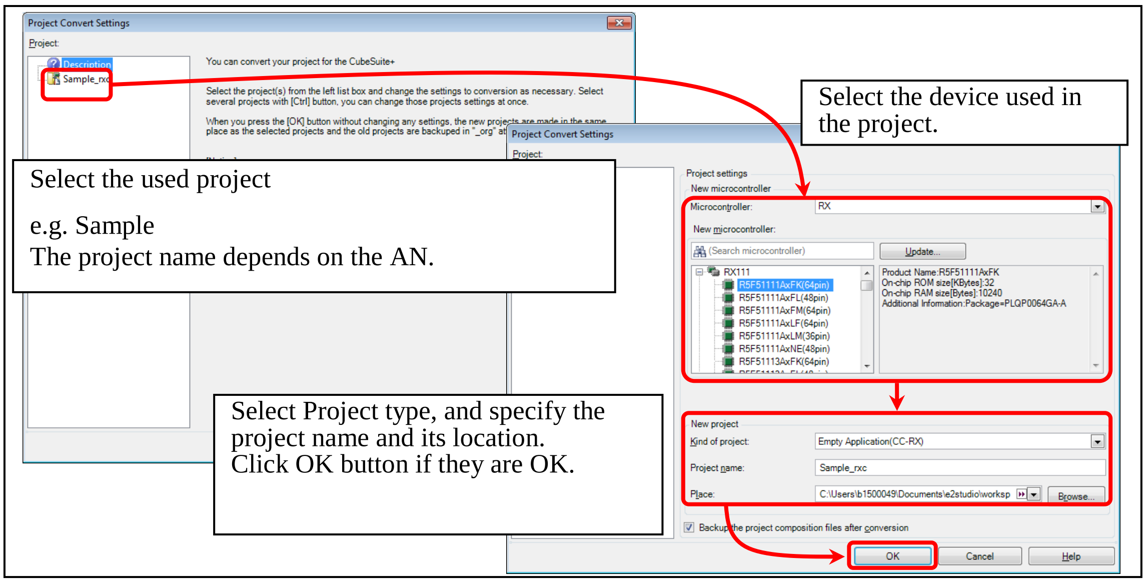


Launch CS+ and click "Start".
Select [Open Exsisting e2studio/CubeSuite/High-performance Embedded Workshop/PM+ project] in Start menu.

Select the file with the extension [.rcpc] and click Open button.

Select [project file for e2studio]

CubeSuite+ - [Start]
File Edit View Project Build Debug Tool Window Help
Start
Project Tree
Loads the project of CubeSuite+. Can also be opened directly from the following link.
GO
Recent Projects
Nothing
Favorite Projects
Nothing
Open Existing e2studio/CubeSuite/High-performance Embedded Workshop/PM+ Project
The project created with e2studio and the old IDE can be converted to the CubeSuite+ project.
GO
Open Project
workspace ▸ Sample_rxc ▸
Search Sample_rxc
Arrange by: Folder
Date modified
Type
.settings 6/30/2013 1:22 PM File folder
HardwareDebug 6/30/2013 1:29 PM File folder
Release 6/30/2013 1:29 PM File folder
src 6/30/2013 1:22 PM File folder
Sample_rxc.rcpc
Libraries
Documents
Music
Pictures
Sample_rxc.rcpc
Project File for e2studio (*.rcpc)
Open
Cancel



Select the device used in the project.

Select the used project

e.g. Sample
The project name depends on the AN.

Select Project type, and specify the project name and its location.
Click OK button if they are OK.

Project Convert Settings
Project:
Description
Sample_rxc
You can convert your project for the CubeSuite+
Select the project(s) from the left list box and change the settings to conversion as necessary. Select several projects with [Ctrl] button, you can change those projects settings at once.
When you press the [OK] button without changing any settings, the new projects are made in the same place as the selected projects and the old projects are backuped in "_org"
Project settings
New microcontroller
Microcontroller: RX
New microcontroller:
(Search microcontroller)
Update...
RX111
R5F51111AxFK(64pin)
R5F51111AxFL(48pin)
R5F51111AxFM(64pin)
R5F51111AxLF(64pin)
R5F51111AxLM(36pin)
R5F51111AxNE(48pin)
R5F51113AxFK(64pin)
Product Name:R5F51111AxFK
On-chip ROM size[KBytes]:32
On-chip RAM size[Bytes]:10240
Additional Information:Package=PLQP0064GA-A
New project
Kind of project: Empty Application(CC-RX)
Project name: Sample_rxc
Place: C:\Users\b1500049\Documents\e2studio\worksp
Browse...
Backup the project composition files after conversion
OK
Cancel
Help

**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 | Summary |
| 1.00 | Mar.18.11 | — | First edition issued |
| 2.00 | Feb.06.13 | — | Revision of the document by firmware upgrade |
| 2.01 | Mar.26.13 | — | Added about IAR edition. |
| 2.10 | Aug.01.13 | — | RL78/L1C, RX111 is supported. Error is fixed. |
| 2.11 | Oct.31.13 | — | 1.4 Folder path fixed.<br>3.2.1 Folder Structure was corrected.<br>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 and TN-RL*-A033B/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