Renesas USB MCU

R01AN2025EJ0110
Rev.1.10
Dec 26, 2014

USB Basic Host and Peripheral Driver using Firmware Integration Technology

Introduction

This application note describes the USB basic firmware, which utilizes Firmware Integration Technology (FIT). This module performs hardware control of USB communication. It is referred to below as the USB-BASIC-F/W FIT module.

Target Device

RX64M Group
RX71M Group

When using this application note with other Renesas MCUs, careful evaluation is recommended after making modifications to comply with the alternate MCU.

Related Documents

1. Universal Serial Bus Revision 2.0 specification
   【http://www.usb.org/developers/docs/】
2. RX64M Group User's Manual: Hardware (Document number .R01UH0377EJ)
3. RX71M Group User's Manual: Hardware (Document number .R01UH0493EJ)

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

Contents

1. Overview

The USB-BASIC-F/W FIT module performs USB hardware control. The USB-BASIC-F/W FIT module operates in combination with sample device class drivers provided by Renesas or device class drivers created by the user. The This module supports the following functions.

<Overall>

- Can operate in either host or peripheral mode.
- Device connect/disconnect, suspend/resume, and USB bus reset processing.
- Control transfer on pipe 0.
- Data transfer on pipes 1 to 9. (bulk or interrupt transfer: CPU access/DTC access)

<Host mode>

In host mode, enumeration as low-speed/full-speed/high-speed device (However, operating speed is different by devices ability.)

- Transfer error determination and transfer retry.
- Multiple device class drivers may be installed without the need to customize USB-BASIC-F/W FIT module.

<Peripheral mode>

- In peripheral mode, enumeration as USB Host of USB1.1/2.0/3.0.

Limitations

This module is subject to the following restrictions.

- The following functions must be provided by the customer to match the system under development.
  - Overcurrent detection processing when connecting USB cables (Host mode).
  - Descriptor analysis (Host mode).
- The pipe usage methods are limited by the pipe information setting functions.
  - The receive pipe uses the transaction counter with the SHTNAK function.
- The structures contain members of different types.
  (Depending on the compiler, the address alignment of the structure members may be shifted.)
- In host mode, USB-BASIC-F/W FIT module does not support suspend/resume of the connected hub and devices connected to the hub's down ports.
- USB-BASIC-F/W FIT module does not support suspend during data transfer. Execute suspend after confirming the data transfer was completed.

Terms and Abbreviations

APL	:	Application program
cstd	:	Prefix of function and file for Peripheral & Host Common Basic F/W
CDP	:	Charging Downstream Port
DCP	:	Dedicated Charging Port
HBC	:	Host Battery Charging control
HCD	:	Host control driver of USB-BASIC-F/W
HDCD	:	Host device class driver (device driver and USB class driver)
hstd	:	Prefix of function and file for Host Basic (USB low level) F/W
HUBCD	:	Hub class sample driver
H/W	:	Renesas USB deviceRenesas USB MCU
MGR	:	Peripheral device state maneger of HCD
PBC	:	Peripheral Battery Charging control
PCD	:	Peripheral control driver of USB-BASIC-F/W
PDCD	:	Peripheral device class driver (device driver and USB class driver)
PP	:	Pre-processed definition
pstd	:	Prefix of function and file for Peripheral Basic (USB low level) F/W
RSK	:	Renesas Starter Kits
STD	:	USB-BASIC-F/W
USB	:	Universal Serial Bus
USB-BASIC-F/W	:	USB Basic Host and Peripheral firmware for Renesas USB MCU
Scheduler	:	Used to schedule functions, like a simplified OS.
Scheduler Macro	:	Used to call a scheduler function (non-OS)
Task	:	Processing unit

1.1 USB-BASIC-F/W FIT module

User needs to integrate this module to the project using r_bsp. User can control USB H/W by using this module API after integrating to the project.
Please refer to the chapter “Adding the Module” about how to add the module.

1.2 Software Configuration

In peripheral mode, USB-BASIC-F/W comprises the peripheral driver (PCD), and the application (APL). PDCD is the class driver and not part of the USB-BASIC-F/W. See Table 1-1. In host mode, USB-BASIC-F/W comprises the host driver (HCD), the manager (MGR), the hub class driver (HUBCD) and the application (APL). HDD and HDCD are not part of the USB-BASIC-F/W, see Table 1-1
The peripheral driver (PCD) and host driver (HCD) initiate hardware control through the hardware access layer according to messages from the various tasks or interrupt handler. They also notify the appropriate task when hardware control ends, of processing results, and of hardware requests.
Manager manages the connection state of USB peripherals and performs enumeration. In addition, manager issues a message to host driver or hub class driver when the application changes the device state. Hub class driver is sample program code for managing the states of devices connected to the down ports of the USB hub and performing enumeration.
The customer will need to make a variety of customizations, for example designating classes, issuing vendor-specific requests, making settings with regard to the communication speed or program capacity, or making individual settings that affect the user interface.

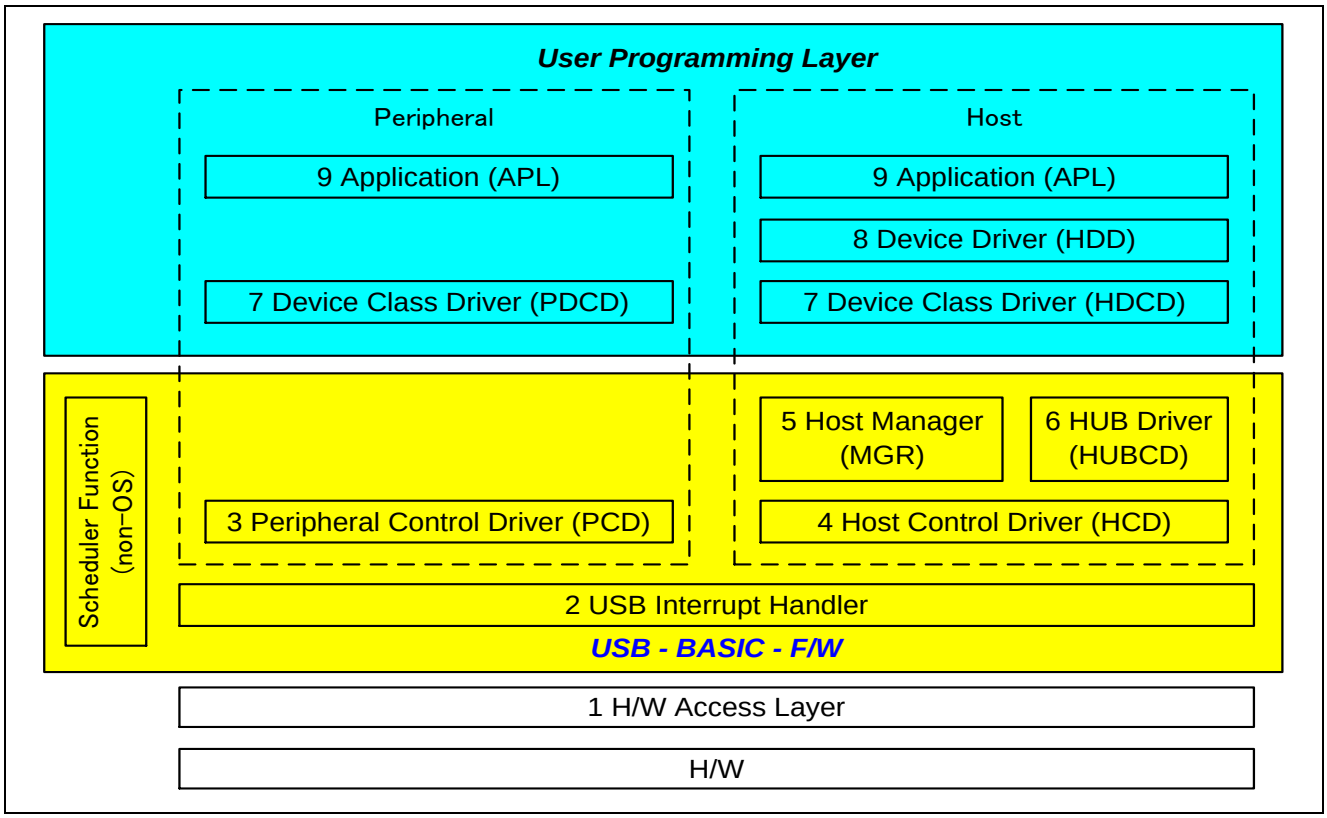

Figure 1-1 Task Configuration of USB-BASIC-F/W

Table 1-1 Software function overview

No	Module Name	Function
1	H/W Access Layer	Hardware control
2	USB Interrupt Handler	USB interrupt handler (USB packet transmit/receive end and special signal detection)
3	Peripheral Control Driver (PCD)	Hardware control in peripheral mode Peripheral transaction management
4	Host control driver (HCD)	Hardware control in host mode Host transaction management
5	Host Manager (MGR)	Device state management Enumeration HCD/HUBCD control message determination
6	HUB Driver (HUBCD)	HUB down port device state management HUB down port enumeration
7	Device Class Driver	Provided by the customer as appropriate for the system.
8	Device Driver	Provided by the customer as appropriate for the system.
9	Application	Provided by the customer as appropriate for the system.

1.3 Scheduler Function and Tasks

When using the non-OS version of the source code, which is set by a macro in r_usb_usrconfig.h, a scheduler function manages requests generated by tasks and hardware according to their relative priority. When multiple task requests are generated with the same priority, they are executed using a FIFO configuration. To assure commonality with non-OS and uITRON-compatible firmware, requests between tasks are implemented by transmitting and receiving messages. In addition, call-back functions are used for responses to tasks indicating the end of a request, so the customer need only install appropriate class drivers for the system and there is no need to modify the scheduler itself. Please refer to “4.3.20 PCD Call-back functions” or “5.5.1 HCD call-backs”.

1.4 Note

1. USB-BASIC-F/W FIT module is not guaranteed to provide USB communication operation. The customer should verify operation when utilizing it in a system and confirm the ability to connect to a variety of different types of devices.
2. The sample program using DMA transfer function is not provided in USB-BASIC-F/W FIT module.The sample program using DTC transfer is provided. Please refer to “9 DTC Transfer” about DTC transfer function.

2. How to Register Class Driver

The class driver which the user created functions as the USB class driver by registering with USB-BASIC-F/W FIT module.

2.1 How to register Peripheral Class Driver

User needs to register the various information for the peripheral class driver which User has developed to USB_PCDREG_t structure. Please refer to"4.4.1 USB_PCDREG_t structure" about USB_PCDREG_t structure.

The following describes how to register user-developed class drivers and applications in the USB-BASIC-F/W FIT module.

```
void usb_papl_registration(USB_UTR_t *ptr)
{
    USB_PCDREG_t    driver;

    /* Driver registration */

    /* Pipe Define Table address */
    driver.pipetbl      = (uint16_t**)&usb_gpvendor_smpl_epptr;
    /* Device descriptor Table address */
    driver.devicetbl    = (uint8_t*)&usb_gpvendor_smpl_DeviceDescriptor;
    /* Qualifier descriptor Table address */
    driver.qualitbl     = (uint8_t*)&usb_gpvendor_smpl_QualifierDescriptor;
    /* Configuration descriptor Table address */
    driver.configtbl    = (uint8_t**)&usb_gpvendor_smpl_ConPtr;
    /* Other configuration descriptor Table address */
    driver.othertbl     = (uint8_t**)&usb_gpvendor_smpl_ConPtrOther;
    /* String descriptor Table address */
    driver.stringtbl    = (uint8_t**)&usb_gpvendor_str_ptr;
    /* Driver init */
    driver.classinit    = &usb_cvendor_dummy_function;
    /* Device default */
/* Condition compilation by the difference of user define */
#if USB_SPEEDSEL_PP == USB_HS_PP
    driver.devdefault   = &usb_pvendor_smpl_DescriptorChange;
#else   /* USB_SPEEDSEL_PP == USB_HS_PP */
    driver.devdefault   = &usb_cvendor_dummy_function;
#endif  /* USB_SPEEDSEL_PP == USB_HS_PP */
    /* Device configured */
    driver.devconfig    = &usb_pvendor_apl_init;
    /* Device detach */
    driver.devdetach    = &usb_pvendor_apl_close;
    /* Device suspend */
    driver.devsuspend   = &usb_cvendor_dummy_function;
    /* Device resume */
    driver.devresume    = &usb_cvendor_dummy_function;
    /* Interfaced change */
    driver.interface    = &usb_cvendor_dummy_function;
    /* Control Transfer */
    driver.ctrltrans    = &usb_cstd_UsrCtrlTransFunction;
    R_usb_pstd_DriverRegistration(ptr, &driver);
}
```

2.2 How to register Host Class Driver

User needs to register the various information for the host class driver which User has developed to USB_HCDREG_t structure. Please refer to"5.6.2 USB_HCDREG_t structure" about USB_HCDREG_t structure.

The following describes how to register user-created class drivers and applications in the USB-BASIC-F/W FIT module.

```
void usb_hapl_registration(USB_UTR_t *ptr)
{
    USB_HCDREG_t    driver;

    /* Driver registration */

    /* Interface Class */
    driver.ifclass      = (uint16_t)USB_IFCLS_VEN;
    /* Target peripheral list */
    driver.tpl          = (uint16_t*)&usb_gapl_devicetpl;
    /* Pipe Define Table address */
    driver.pipetbl      = (uint16_t*)&usb_shvendor_smpl_def_eptbl;
    /* Driver init */
    driver.classinit    = &usb_cvendor_dummy_function;
    /* Driver check */
    driver.classcheck   = &usb_hvendor_class_check;
    /* Device configured */
    driver.devconfig    = &usb_hvendor_apl_init;
    /* Device detach */
    driver.devdetach    = &usb_hvendor_apl_close;
    /* Device suspend */
    driver.devsuspend   = &usb_cvendor_dummy_function;
    /* Device resume */
    driver.devresume    = &usb_cvendor_dummy_function;

    R_usb_hstd_DriverRegistration(ptr, &driver);
}
```

3. Task ID etc and Priority Setting

Define the setting value for Device class task ID and User task ID etc within the following range. Please use R_usb_cstd_SetTaskPri(API) about Task priority setting.

Task ID	:	4 to (USB_IDMAX - 1)
Mailbox ID	:	Set the same value as Task ID
Memory pool ID	:	Set the same value as Task ID
Task Priority	:	4 to (USB_PRIMAX - 1)

[Note]

1. USB_IDMAX and USB_PRIMAX is the defined by User in r_usb_basic_config.h file.
2. Please define ID number for User task ID etc in the application file.

4. Peripheral

4.1 Peripheral Control Driver (PCD)

4.1.1 Basic functions

PCD is a program for controlling the hardware. PCD analyzes requests from PDCD (not part of the USB-BASIC-F/W FIT module) and controls the hardware accordingly. It also sends notification of control results using a user provided call-back function. PCD also analyzes requests from hardware and notifies PDCD accordingly.
PCD accomplishes the following:

1. Control transfers. (Control Read, Control Write, and control commands without data stage.)
2. Data transfers. (Bulk, interrupt) and result notification.
3. Data transfer suspensions. (All pipes.)
4. USB bus reset signal detection and reset handshake result notifications.
5. Suspend/resume detections.
6. Attach/detach detection using the VBUS interrupt.
7. Hardware control when entering and returning from the clock stopped (low-power sleep mode) state.

4.1.2 Issuing requests to PCD

API functions are used when hardware control requests are issued to the PCD and when performing data transfers. API functions and library functions can be used to obtain information for managing PCD.
In response to a request from an upper layer task, PCD sends a result notification by means of a call-back function.
PCD has no API functions for class/vendor requests.

4.1.3 USB requests

The following standard requests are supported by PCD.

GET_STATUS
GET_DESCRIPTOR
GET_CONFIGURATION
GET_INTERFACE
CLEAR_FEATURE
SET_FEATURE
SET_ADDRESS
SET_CONFIGURATION
SET_INTERFACE

PCD answers requests other than the above with a STALL response.
When the PCD receives one of the above standard requests, it executes the functions listed in Table4-1 according to the control transfer stage.
Note that, if the received USB request is a device class request or a vender class request, the user needs to create a program to judge these requests and register the program in the driver.

The following shows the peripheral processes available for the control transfer stage, based on the standard request received from a host.

ControlRead	:	When a valid request is received, data for transmission to the host is generated, and written to the FIFO
ControlWrite	:	When a valid request is received, enables data reception from the host.
NoDataControl	:	When a valid request is received, returns the status.
ControlRead-Status	:	Receives the status from the host.
ControlWrite-Status	:	Returns the status to the host.
Control transfer end	:	Sends notification to PDCD of a request received from host.

Table4-1 Peripheral Function Used

Control Transfer Stage	Device Class	Response to Host
ControlRead	usb_pstd_StandReq1()	Transmit data
ControlWrite	usb_pstd_StandReq2()	Receive data
NoDataControl	usb_pstd_StandReq3()	Return status
ControlRead -Status	usb_pstd_StandReq4()	Receive status
ControlWrite -Status	usb_pstd_StandReq5()	Return status
Control transfer end	usb_pstd_StandReq0()	None

4.2 API Information

This Driver API follows the Renesas API naming standards.

4.2.1 Hardware Requirements

This driver requires your MCU support the following features:

- USB

4.2.2 Software Requirements

This driver is dependent upon the following packages:

- r_bsp
- r_usb_dtc (using DTC transfer)

4.2.3 Supported Toolchains

This driver is tested and working with the following toolchains:

- Renesas RX Toolchain v.2.01.00b

4.2.4 Header Files

All API calls and their supporting interface definitions are located in r_usb_basic_if.h.

4.2.5 Integer Types

This project uses ANSI C99 “Exact width integer types” in order to make the code clearer and more portable. These types are defined in *stdint.h*.

4.2.6 Compile Setting

The configuration option settings for this module are specified in r_usb_basic_config.h.
The option names and setting values are described below.

Configuration options in r_usb_basic_config.h	
USB_DTC_ENABLE	Selects whether or not the DTC FIT module is used. = Enabled: DTC FIT module is used. = Disabled: DTC FIT module is not used.
USB_FUNCSEL_USBIP0_PP	Specifies the operating mode of the USB module (USBb). = USB_PERI_PP: USB module operates as a peripheral. = USB_NOUSE_PP: USB module does not operate.
USB_FUNCSEL_USBIP1_PP	Specifies the operating mode of the USB module (USBA/USBAa). = USB_PERI_PP: USB module operates as a peripheral. = USB_NOUSE_PP: USB module does not operate.
USB_SPEED_PP	Specifies the speed mode of the USB module (USBA/USBAa). = USB_HS_PP: Use Hi-speed (RX71M only) = USB_FS_PP: Use Full-speed
USB_PERI_BC_ENABLE	Specifies whether or not battery charging functionality is enabled in Peripheral mode. = Enabled: Battery charging functionality is used. = Disabled: Battery charging functionality is not used.
USB_CPUBYTE_PP	Specifies the endian mode for USB FIFO register access. The setting of this item should match the endian mode of the CPU. = USB_BYTE_LITTLE_PP: Little-endian = USB_BYTE_BIG_PP: Big-endian
USB_CPU_LPW_PP	Specifies whether or not low-power mode is used. = USB_LPWR_USE_PP: Low-power mode is used. = USB_LPWR_NOT_USE_PP: Low-power mode is not used.
USB_STRINGNUM	Specifies the number of the string descriptor.

4.2.7 Argument

The structure for the arguments of the API functions is shown below. For details, please refer to USB Basic Firmware application note (Document No.R01AN2025EJ).

```
typedef struct USB_UTR
{
  USB_MH_t      msghead;              /* Message header (for RTOS) / No used */
  uint16_t      msginfo;              /* Message information */
  uint16_t      keyword;              /* Sub information(Port No/Pipe No etc)  */
  union {
        USB_REGADR_t      ipp;      /* USB IP address (for USBb module)*/
        USB_REGADR1_t     ipp1;     /* USB IP address (for USBA/USBAa module) */
  };
  uint16_t      ip;                   /* USB IP number*/
  uint16_t      result;               /* USB transfer result */
  USB_CB_t      complete;             /* Pointer to the callback function */
  void          *tranadr;             /* Pointer to USB transfer buffer */
  uint32_t      tranlen;              /* USB data length*/
  uint16_t      *setup;               /* Pointer to Setup packet data */
  uint16_t      status;               /* USB status */
  uint16_t      pipectr;              /* PIPECTR register */
  uint8_t       errcnt;               /* Error count for transferring */
  uint8_t       segment;              /* Segment information */
  void          *usr_data;            /* Other various information */
} USB_UTR_t;
```

4.2.8 Adding the Module

This module must be added to an existing e^2 studio project. By using the e^2 studio plug-in, it is possible to update the include file path automatically. It is therefore recommended that this plug-in be used to add the project.

For instructions when using e^2 studio, refer to RX Family: Integration into e^2 studio, Firmware Integration Technology (document No. R01AN1723EU).

4.3 API (Application Programming Interface)

A PDCD module (USB class driver) implements hardware control requests by using the PCD API.
The return values of each API function are scheduler macro error codes. A list of PCD API functions is shown in Table4-2.

Table4-2 List of PCD API Functions

Function	Description
R_USB_Open()	Start the USB module
R_USB_Close()	Stop the USB module
R_USB_GetVersion()	Return the driver version
R_usb_pstd_PcdOpen()	Start the PCD task
R_usb_pstd_PcdClose()	End the PCD task
R_usb_pstd_TransferStart()	Data transfer execution request
R_usb_pstd_TransferEnd()	Data transfer forced end request
R_usb_pstd_ChangeDeviceState()	USB device state change request
R_usb_pstd_DeviceInformation()	Get USB device state information
R_usb_pstd_DriverRegistration()	Register the PDCD
R_usb_pstd_PcdTask()	The PCD Task
R_usb_pstd_ControlRead()	FIFO access execution request for control read transfer
R_usb_pstd_ControlWrite()	FIFO access execution request for control write transfer
R_usb_pstd_ControlEnd()	Control transfer end request
R_usb_pstd_SetStall()	Set STALL to PID of PIPE that is specified by argument and the callback function is called
R_usb_pstd_SetPipeStall()	Set STALL to PID of PIPE that is specified by argument
R_usb_cstd_GetUsbIpAdr()	Get USB register base address
R_usb_cstd_UsbIpInit()	Initialize USB module
R_usb_cstd_ClearHwFunction()	USB-Related Register Initialization Request

4.3.1 R_USB_Open

Power on USB module.

Format

usb_err_t R_USB_Open(usb_ip_t ip_type)

Arguments

ip_type IP number (USB_IP0/USB_IP1) of USB module.

Return Value

USB_SUCCESS	Success
USB_ERR_BUSY	Locked USB module by other S/W module
USB_ERR_OPENED	USB module currently in operation

Description

This function applies power to the USB module specified in the argument. USBb module is powered on when specifing “USB_IP0” in the argument, USBA/USBAa module is powered on when specifing “USB_IP1” in the argument.

Note

1. Don’t call this API after powering on USB module.

Example

```
void  usb_smp_task()
{
  USB_ER_t    err;
        :
  err = R_USB_Open(USB_IP0);        /* Start USBb module */
  if(err != USB_SUCCESS)
  {
    /* error */
  }
        :
}
```

4.3.2 R_USB_Close

Power off USB module.

Format

usb_err_t R_USB_Close(usb_ip_t ip_type)

Arguments

ip_type IP number (USB_IP0/USB_IP1) of USB module.

Return Value

USB_SUCCESS	Success
USB_ERR_BUSY	Locked USB module by other S/W module.
USB_ERR_NOT_OPEN	USB module is not opened.

Description

This function removes power to the USB module specified in the argument. USBb module is powered off when specifing “USB_IP0” in the argument, USBA/USBAa module is powered off when specifing “USB_IP1” in the argument.

Note

--

Example

```
void  usb_smp_task()
{
  USB_ER_t    err;
        :
  err = R_USB_Close(USB_IP0);         /* Stop USBb module */
  if(err != USB_SUCCESS)
  {
    /* error */
  }
        :
}
```

4.3.3 R_USB_ GetVersion

Return the driver version number

Format

uint32_t R_USB_GetVersion(void)

Arguments

— —

Return Value

Version number

Description

This function returns the driver version number at runtime

Note

--

Example

```
void  usb_smp_task()
{
  uint32_t   version;
       :
  version = R_USB_GetVersion();
       :
}
```

4.3.4 R_usb_pstd_PcdOpen

Start PCD task

Format

USB_ER_t R_usb_pstd_PcdOpen(USB_UTR_t *ptr)

Arguments

*ptr Pointer to a USB transfer structure

Return Value

USB_E_OK Success

Description

This function initializes the pipe information.
Return value is always USB_E_OK.

Note

1. The user application should register the PDCD in the PCD and then call this function during initialization. Please register the PDCD by using the function R_usb_pstd_DriverRegistration.
2. Please do not call this function after starting the PCD task

Example

```
void  usb_smp_task()
{
  USB_UTR_t  *ptr;
    :
  R_usb_pstd_PcdOpen(ptr);
    :
}
```

4.3.5 R_usb_pstd_PcdClose

End PCD task

Format

USB_ER_t R_usb_pstd_PcdClose(USB_UTR_t *ptr)

Arguments

*ptr Pointer to a USB Transfer Structure

Return Value

USB_E_OK Success

Description

No processing.
Return value is always USB_E_OK.

Note

1. Besides above arguments, also set the following members of the USB Transfer Structure
   USB_REGADR_t ipp :USB register base address
   uint16_t ip :USB IP number
2. Call this function from the user application or class driver.
3. Please do not call this function after starting PCD task.

Example

```
void  usb_smp_task()
{
  USB_UTR_t  *ptr;
    :
  R_usb_pstd_PcdClose(ptr);
    :
}
```

4.3.6 R_usb_pstd_TransferStart

Data transfer request

Format

USB_ER_t R_usb_pstd_TransferStart(USB_UTR_t *ptr)

Arguments

*ptr	Pointer to a USB Transfer Structure

Return Value

USB_E_OK	Success
USB_E_ERROR	Failure

Description

This function transfers data via the pipe.
This function executes data transfer requests to the PCD. After receiving the request, PCD executes the data transfer processing based on the transfer information stored in the USB Transfer Structure. Besides above arguments, also set the following information of this structure:

USB_MH_t	msghead	:	Set to NULL
uint16_t	keyword	:	Pipe number
void*	tranadr	:	Start address of the data buffer
uint32_t	tranlen	:	Data transfer size
uint16_t	setup	:	Set to 0 (zero)
USB_CB_t	complete	:	Pointer to call-back function
uint8_t	segment	:	Status
USB_REGADR_t	ipp	:	USB register base address
uint16_t	ip	:	USB IP number

When data transfer ends (specified data size reached, short packet received, error occurred), the Transfer Structure's call-back function is executed. The remaining transmit/receive data length, status, error count and transfer end is set as the parameters of this call-back function. Please refer to "Table4-3 R_usb_pstd_TransferStart call-back" about the call-back function.

Note

1. Call this function from the user application or class driver.
2. The structure members indicated by the argument include pipe number, transfer data start address, transfer data length, status, call-back function at end, etc.
3. When the received data is n times of the maximum packet size and less than the expected received data length, it is considered that the data transfer is not ended and a callback is not generated.

Example

```
USB_UTR_t   trn_msg[USB_NUM_USBIP][USB_MAXPIPE_NUM + 1];
USB_ER_t usb_smp_task(USB_UTR_t *ptr, uint16_t pipe, uint32_t size, uint8_t *table)
{
    :
  /* Transfer information setting */
  trn_msg[ptr->ip][pipe].msghead  = (USB_MH_t)NULL;   /* NULL only */
  trn_msg[ptr->ip][pipe].keyword  = pipe;  /* Pipe No */
  trn_msg[ptr->ip][pipe].tranadr  = table; /* Pointer to data buffer */
  trn_msg[ptr->ip][pipe].tranlen  = size;  /* Transfer size */
  trn_msg[ptr->ip][pipe].setup    = 0;
  trn_msg[ptr->ip][pipe].complete = ptr->complete;  /* Call-back function */
  trn_msg[ptr->ip][pipe].segment  = USB_TRAN_END;  /* status */
  trn_msg[ptr->ip][pipe].ipp      = ptr->ipp;  /* USB IP base address */
  trn_msg[ptr->ip][pipe].ip       = ptr->ip;   /* USB IP No */

  /* Data transfer request */
  err = R_usb_pstd_TransferStart((USB_UTR_t *)&trn_Msg [ptr->ip][pipe]);

  return err;
    :
}
```

4.3.7 R_usb_pstd_TransferEnd

Data transfer forced end request

Format

USB_ER_t R_usb_pstd_TransferEnd(USB_UTR_t *ptr, uint16_t pipe, uint16_t status)

Arguments

*ptr	Pointer to a USB transfer structure
pipe	Pipe number
status	USB communication status

Return Value

USB_E_OK	Success
USB_E_ERROR	Failure

Description

This function forces data transfer via the specified pipe to end.

The function executes a data transfer forced end request to the PCD. After receiving the request, the PCD executes the data transfer forced end request processing.

When a data transfer is forcibly ended, the function calls the call-back function set in “R_usb_pstd_TransferStart” at the time the data transfer was requested. The information of the remaining transmit/receive data length, status, error count and transfer end is set in the parameter of this call-back.

Note

1. Besides above arguments, also set the following members of the USB Transfer Structure.
   USB_REGADR_t ipp :USB register base address
   uint16_t ip :USB IP number
2. Call this function from the user application or class driver.

Example

```
void  usb_smp_task(USB_UTR_t *ptr)
{
  uint16_t status;
  uint16_t pipe;
    :
  pipe    = USB_PIPE1;
  status  = USB_DATA_STOP;

  /* Transfer end request */
  err = R_usb_pstd_TransferEnd(ptr, pipe, status);

  return err;
    :
}
```

4.3.8 R_usb_pstd_ ChangeDeviceState

USB peripheral device state change request

Format

USB_ER_t R_usb_pstd_ChangeDeviceState(USB_UTR_t *ptr, uint16_t msginfo, uint16_t member, USB_CB_INFO_t complete)

Arguments

*ptr	Pointer to a USB Transfer Structure
state	Device state to be transitioned to
port_no	Port number
complete	Call-back function executed at end of PCD processing

Return Value

USB_E_OK	Success
USB_E_ERROR	Failure

Description

This function sends a request to the PCD to change the USB device state by setting one of the following values of argument “state”, and then call the ‘complete’ call-back function.

- USB_DO_REMOTEWAKEUP
  Remote wakeup execution request to PCD
- USB_DP_ENABLE
  D+ line pull-up request to PCD
- USB_DP_DISABLE
  D+ line pull-up cancel request to PCD
- USB_DO_STALL
  STALL response execution request to PCD

Note

1. Besides above arguments, also set the following members of the USB Transfer Structure.
   USB_REGADR_t ipp :USB register base address
   uint16_t ip :USB IP number
2. Call this function from the user application or class driver.
3. If connected/disconnected interrupt is detected, a D+ line pull-up cancel is executed automatically by firmware.

Example

```
void  usb_smp_task(USB_UTR_t *ptr)
{
    :
  /* STALL response request */
  R_usb_pstd_ChangeDeviceState(ptr, USB_DO_STALL, USB_PIPE0, usb_smp_dummy)
    :
}
```

4.3.9 R_usb_pstd_DeviceInformation

Get a USB peripheral's device state information

Format

void R_usb_pstd_DeviceInformation (USB_UTR_t *ptr, uint16_t *tbl)

Arguments

*ptr	Pointer to a USB Transfer Structure
*tbl	Pointer to the buffer that the device information is stored.

Return Value

— —

Description

This function gets USB peripherals device state information. It stores the following information at the address designated by the argument "tbl".

[0] : USB device state

DVSQ and VBSTS bit value in he interrupt status register (INTSTS0) is stored.

[1] : USB transfer speed

USB_HSCONNECT (0xC0)	: High Speed
USB_FSCONNECT (0x80)	: Full Speed
USB_LSCONNECT (0x40)	: Low Speed
USB_NOCONNECT (0x00)	: Powerd state or non connect

[2] : Configuration number used

[3] : Number of Interface

[4] : Remote Wakeup Flag

Note

1. Besides above arguments, also set the following members of the USB Transfer Structure before calling this function.

USB_REGADR_t	ipp	:USB register base address
uint16_t	ip	:USB IP number

2. Call this function from the user application or class driver.

Example

```
void  usb_smp_task(USB_UTR_t *ptr)
{
  uint16_t  res[5];
    :
  /* Get USB Device Information */
  R_usb_pstd_DeviceInformation(ptr, (uint16_t *)res);
    :
}
```

4.3.10 R_usb_pstd_DeviceRegistration

Register a PDCD

Format

void	R_usb_pstd_DriverRegistration(USB_UTR_t *ptr, USB_PCDREG_t *registinfo)

Arguments

*ptr	Pointer to a USB Transfer Structure
*registinfo	Pointer to class driver structure

Return Value

—	—

Description

This function registers the PDCD information, which is registered in the class driver structure, in the PCD. After registration is complete, the initialization call-back function is executed.

Note

1. Besides above arguments, also set the following members of the USB Transfer Structure before calling this function:
   USB_REGADR_t ipp :USB register base address
   uint16_t ip :USB IP number
2. The user must call this function from the user's program during initialization.
3. Only one device can be registered. Please refer to '' about the registration information.

Example

```
void  usb_smp_registration(USB_UTR_t *ptr)
{
  USB_PCDREG_t  driver;
    :
  /* Register PDCD information to driver */
  /* Pipe Define Table address */
  driver.pipetbl   = (uint16_t**)&usb_gpvendor_smpl_epptr;
  /* Device descriptor Table address */
  driver.devicetbl = (uint8_t*)&usb_gpvendor_smpl_DeviceDescriptor;
  /* Qualifier descriptor Table address */
  driver.qualitbl  = (uint8_t*)&usb_gpvendor_smpl_QualifierDescriptor;
  /* Configuration descriptor Table address */
  driver.configtbl = (uint8_t**)&usb_gpvendor_smpl_ConPtr;
  /* Other configuration descriptor Table address */
  driver.othertbl  = (uint8_t**)&usb_gpvendor_smpl_ConPtrOther;
  /* String descriptor Table address */
  driver.stringtbl = (uint8_t**)&usb_gpvendor_str_ptr;
  /* Driver init */
  driver.classinit = &usb_cvendor_dummy_function;

  R_usb_pstd_DriverRegistration(ptr, &driver);
}
```

4.3.11 R_usb_pstd_PcdTask

The PCD Task

Format

void	R_usb_pstd_PcdTask(USB_VP_INT stacd)

Arguments

stacd	Task start code Not used

Return Value

—	—

Description

This function executes hardware control when running as a USB peripheral.

Note

1. Call this in the loop that executes the scheduler processing

```
void  usb_smp_mainloop(void)
{
  while(1)
  {
  /* Start non-OS scheduler */
  R_usb_cstd_Scheduler();
  /* Flag checking */
  if(USB_FLGSET == R_usb_cstd_CheckSchedule())
  {
        :
   R_usb_pstd_PcdTask((USB_VP_INT)0);
        :
  }
    :
}
```

4.3.12 R_usb_pstd_ControlRead

FIFO access request for control read transfer

Format

uint16_t R_usb_pstd_ControlRead(USB_UTR_t *ptr, uint32_t bsize, uint8_t *table)

Arguments

*ptr	Pointer to a USB transfer structure, containing pipe number, read size, etc
bsize	Transmit data buffer size
*table	Pointer to transmit data buffer address

Return Value

end_flag	Data transfer result

Description

During a control IN transfer, this function issues a FIFO access execution request to PCD during the data stage.

PCD reads data from the area referenced by the argument (table) and writes it to the FIFO buffer of the hardware.

This function returns the following result as return value.

- USB_WRITESHRT
  Data write end (short packet data write)
- USB_WRITEEND
  Data write end (no additional data/transmission of packet with data length 0）
- USB_WRITING
  Data write in progress (additional data present)
- USB_FIFOERROR
  FIFO access error

Note

1. Besides above arguments, also set the following members of the USB Transfer Structure before calling this function.
   USB_REGADR_t ipp :USB register base address
   uint16_t ip :USB IP number
2. Please call this function at the control IN transfer data stage. Please refer to4.6 Peripheral **Control Transfer**

Example

```
uint8_t  usb_smp_buf;
void  usb_smp_task(USB_UTR_t *ptr)
{
    :
  R_usb_pstd_ControlRead(ptr, (uint32_t)1, (uint8_t*)(& usb_smp_buf));
    :
}
```

4.3.13 R_usb_pstd_ControlWrite

FIFO access request for control write transfer

Format

void	R_usb_pstd_ControlWrite(USB_UTR_t *ptr, uint32_t bsize, uint8_t *table)

Argument

*ptr	Pointer to a USB transfer structure containing pipe number etc
bsize	Receive data buffer size for control write transfer
*table	Receive data buffer address for control write transfer

Return Value

—	—

Description

During a control transfer data stage, this function issues a FIFO access execution request to PCD.

PCD reads data from the FIFO buffer of the hardware and writes it to the area referenced by the argument table.

Note

1. Besides above arguments, also set the following members of the USB Transfer Structure.
   USB_REGADR_t ipp :USB register base address
   uint16_t ip :USB IP number
2. Please call this function at the control OUT transfer data stage. Please refer to '4.6 Peripheral Control Transfer'.

Example

```
uint8_t  usb_smp_buf;
void  usb_smp_task(USB_UTR_t *ptr)
{
    :
  R_usb_pstd_ControlWrite(ptr, (uint32_t)1, (uint8_t*)(& usb_smp_buf));
    :
}
```

4.3.14 R_usb_pstd_ControlEnd

Control transfer end request

Format

void	R_usb_pstd_ControlEnd(USB_UTR_t *ptr, uint16_t status)

Argument

*ptr	Pointer to a USB Transfer Structure
status	Status

Return Value

—	—

Description

This function issues a request for control transfer status stage execution to PCD. It is called at the control transfer status stage.

Besides above arguments, also set the following value for the status argument.

- USB_CTRL_END
  Status stage normal end.
- USB_DATA_STOP
  Return NAK to host at status stage.
- USB_DATA_OVR
  Return STALL to host at status stage.

Note

1. Besides above arguments, also set the following members of the USB Transfer Structure.
   USB_REGADR_t ipp :USB register base address
   uint16_t ip :USB IP number
2. Please call this function at the control transfer status stage. Refer to 4.6 Peripheral **Control Transfer**.

Example

```
void  usb_smp_task(USB_UTR_t *ptr)
{
    :
  R_usb_pstd_ControlEnd(ptr, (uint16_t)USB_CTRL_END);
    :
}
```

4.3.15 R_usb_pstd_SetStall

Set STALL to PID of PIPE with callback

Format

void	R_usb_pstd_SetStall (USB_UTR_t *ptr, USB_CB_t complete, uint16_t pipeno)

Argument

*ptr	Pointer to a USB Transfer Structure
complete	Callback function
pipeno	Pipe number

Return Value

—	—

Description

Set STALL as PID of the pipe number specified by the argument. The callback function that is specified by argument *complete* will be called when a stall packet has been sent.

Note

1. Besides above arguments, also set the following members of the USB Transfer Structure.
   USB_REGADR_t ipp :USB register base address
   uint16_t ip :USB IP number
2. Call this function from the user application, or the class driver.

Example

```
void  usb_smp_task(USB_UTR_t *ptr)
{
    :
  R_usb_pstd_SetStall( ptr, (USB_CB_t)setstall_cb, pipeno );
    :
}
```

4.3.16 R_usb_pstd_SetPipeStall

Set STALL to PID of PIPE (no callback)

Format

void	R_usb_pstd_SetPipeStall (USB_UTR_t *ptr, uint16_t pipeno)

Argument

*ptr	Pointer to a USB Transfer Structure
pipeno	Pipe number

Return Value

—	—

Description

Setting STALL as PID for the specified pipe number

Note

1. Besides above arguments, also set the following members of the USB Transfer Structure.

USB_REGADR_t	ipp	:USB register base address
uint16_t	ip	:USB IP number

2. Please call this function from the user application or the device class.

Example

```
void  usb_smp_task(USB_UTR_t *ptr)
{
    :
  R_usb_pstd_SetPipeStall( ptr, pipeno );
    :
}
```

4.3.17 R_usb_pstd_GetUsbIpAdr

Get the USB register base address

Format

void R_usb_cstd_GetUsbIpAdr(uint16_t usbip)

Argument

usbip USB IP number

Return Value

USB register base address

Description

Return USB register base address of the specified USB IP.

Note

1. Please call this function from the user application.

Example

```
void  usb_smp_task( void )
{
    USB_UTR_t  utr;
               :
    utr.ip = USB_HOST_USBIP_NUM;
    utr.ipp = R_usb_cstd_GetUsbIpAdr(USB_HOST_USBIP_NUM );
               :
}
```

4.3.18 R_usb_cstd_UsbIpInit

Initialize the USB module

Format

void R_usb_cstd_UsbIpInit(USB_UTR_t *ptr, uint16_t mode)

Argument

*ptr	Pointer to a USB Transfer Structure
mode	USB operation mode (USB_HOST, USB_PERI)

Return Value

—

Description

This API initializes USB module beased on the second argument.

USB_HOST	: Host mode setting
USB_PERI	: Peripheral mode setting

Note

1. Besides above arguments, also set the following members of the USB Transfer Structure.

USB_REGADR_t	ipp	: USB register base address
uint16_t	ip	: USB IP number

2. The user must call this function from the user application during initialization.

Example

```
void  usb_smp_init( void )
{
        :
    USB_UTR_t  utr;
       :
    utr.ip = USB_HOST_USBIP_NUM;
    utr.ipp = R_usb_cstd_GetUsbIpAdr(USB_HOST_USBIP_NUM );
       :
    R_usb_cstd_UsbIpInit( &utr );
       :

}
```

4.3.19 R_usb_pstd_ ClearHwFunction

USB H/W register initialization

Format

void R_usb_cstd_ClearHwFunction(USB_UTR_t *ptr)

Argument

*ptr Pointer to a USB Transfer Structure

Return Value

— —

Description

USB H/W register initialization request

Note

1. Besides above arguments, also set the following members of the USB Transfer Structure.
   USB_REGADR_t ipp :USB register base address
   uint16_t ip :USB IP number

4.3.20 PCD Call-back functions

Table4-3 R_usb_pstd_TransferStart call-back

Call-back	Data Transfer Request call-back function		
Call format	(*USB_CB_t)(USB_UTR_t*);		
Arguments	USB_UTR_t*		Submitted USB_UTR_t pointer
Return values	—		—
Description	This function is executed at data transfer* end. (A transfer end condition at the end of a data transfer, of the size specified by the transfer application, when a short packet is received.). The remaining transmit/receive data length and the error counts are updated.		
Notes	Bulk/Interrupt data transfer		

Table4-4 Call-back of control transfer

Call-back	Call-back function of control transfer without standard request		
Call format	(*USB_CB_TRN_t)(USB_UTR_t *, USB_REQUEST_t *, uint16_t)		
Arguments	USB_UTR_t*		Pointer to a USB Transfer Structure
	USB_REQUEST_t *		Information of request
	uint16_t		Information of stage
Return values	—	—	—
Description	This call-back is called by USB interrupt in control transfer without standard request.		
Notes			

Table4-5 Other call-backs

Call-back	Other call-back Functions		
Call format	(*USB_CB_INFO_t)(USB_UTR_t *,uint16_t, uint16_t)		
Arguments	USB_UTR_t*		Pointer to a USB Transfer Structure
	uint16_t		NOARGUMENT:Not used
	uint16_t		NOARGUMENT:Not used
Return values	—	—	—
Description	usb_pstd_Remote: Executed at end of remote wakeup processing. usb_pstd_ClearSeqbit: Executed when the sequence toggle bit is cleared. R_usb_pstd_SetStall: Executed when the sequence toggle bit is set to the stall state. R_usb_pstd_TransferEnd: Executed at data transfer end.		
Notes			

4.4 Structure Definitions

The structures used in USB peripheral mode are described below. They are defined in file *usb_ctypedef.h*.

4.4.1 USB_PCDREG_t structure

The USB_PCDREG_t structure is used to register PDCD information using the function *R_usb_pstd_DriverRegsitration*. The call-back function registered in USB_PCDREG_t is executed when the device state changes, etc.
Table4-6 lists the members of the USB_PCDREG_t structure.

Table4-6 Member of USB_PCDREG_t structure

Type	Member	Description
uint16_t	**pipetbl	Register the address of the pipe information table.
uint8_t	*devicetbl	Register the device descriptor address.
uint8_t	*qualitbl	Register the device qualifier descriptor address.
uint8_t	**configtbl	Register the address of the configuration descriptor address table.
uint8_t	**othertbl	Register the address of the other speed descriptor address table.
uint8_t	**stringtbl	Register the address of the string descriptor address table.
USB_CB_INFO_t	classinit	Register the function to be started when initializing PDCD. It is called at registration.
USB_CB_INFO_t	devdefault	Register the function to be started when transitioning to the default state. It is called when a USB reset is detected.
USB_CB_INFO_t	devconfig	Register the function to be started when transitioning to the configured state. It is called in the SET_CONFIGURATION request status stage.
USB_CB_INFO_t	devdetach	Register the function to be started when transitioning to the detach state. It is called when a detached condition is detected.
USB_CB_INFO_t	devsuspend	Register the function to be started when transitioning to the suspend state. It is called when a suspend condition is detected.
USB_CB_INFO_t	devresume	Register the function to be started when transitioning to the resume state. It is called when a resume condition is detected.
USB_CB_INFO_t	interface	Register the function to be started when an interface change occurs. It is called in the SET_INTERFACE request status stage.
USB_CB_TRN_t	ctrltrans	Register the function to be started when a user-initiated control transfer occurs.

4.4.2 The USB_REQUEST structure

USB_REQUEST_t is the structure where the latest USB request other than the standard request is stored.
This structure is used as the argument to the call-back function that was registered in member *ctrltrans* of the USB_PCDREG_t structure.
Table4-7 shows the member of USB_REQUEST_t structure.

Table4-7 Member of USB_REQUEST_t structure

Type	Member	Description
uint16_t	ReqType	The value is bmRequestType[D7-D0] of request. (The value is BMREQUESTTYPE of USBREQ register.) When this value is used, mask with USBC_BMREQUESTTYPE(0x00FFu).
uint16_t	ReqTypeType	The value is Type of bmRequestType[D6-D5] of request. (The value is b6-5 of BMREQUESTTYPE of USBREQ register.) When this value is used, mask with USBC_BMREQUESTTYPETYPE(0x0060u)
uint16_t	ReqTypeRecip	The value is Recipient of bmRequestType[D4-D0] of request. (The value is b4-0 of BMREQUESTTYPE of USBREQ register.) When this value is used, mask with USBC_BMREQUESTTYPERECIP(0x001Fu)
uint16_t	ReqRequest	The value is bRequest of request. (The value is BREQUEST of USBREQ register.) When this value is used, mask with USBC_BREQUEST(0xFF00u)
uint16_t	ReqValue	The value is wValue of request. (The value is USBVAL register.)
uint16_t	ReqIndex	The value is wIndex of request. (The value is USBINDEX register.)
uint16_t	ReqLength	The value is wLength of request. (The value is USBLENG register.)

4.5 Pipe Information Table

In order for the PDCD to perform data communication on pipes 1 to 9, it must maintain information tables to define pipe settings. A pipe information table comprises the necessary register setting values for performing data transfers and a specification of the FIFO port transfer direction (CPU transfer or DTC transfer).

The follow shows the sample of the pipe information table.

```
uint16_t usb_gvendor_smpl_eptbl[] =
{
    USB_PIPE1,                                          ←Pipe definition item 1
    USB_NONE | USB_BFREOFF | USB_DBLBOFF |
    USB_CNTMDOFF | USB_SHTNAKOFF | USB_EP1,             ←Pipe definition item 2
    USB_BUF_SIZE(1024) | USB_BUF_NUMB(8),               ←Pipe definition item 3
    64,                                                 ←Pipe definition item 4
    USB_NONE,                                           ←Pipe definition item 5
    USB_CUSE,                                           ←Pipe definition item 6
        :
        :
    USB_PDTBEND,
}
```

A pipe information table comprises the following six items (uint16_t × 6).

1. Pipe Window Select register (address 0x64)
2. Pipe Configuration register (address 0x68)
3. Pipe Buffer Designation register (address 0x6A)
4. Pipe Maximum Packet Size register (address 0x6C)
5. Pipe Period Control register (address 0x6E)
6. FIFO port usage method

(1). Pipe definition item 1

Specify the value to be set in the pipe window select register.

Pipe select	:	Specify the selected pipe (USB_PIPE1 to USB_PIPE9)

Restrictions

—

(2). Pipe definition item 2

Specify the values to be set in the pipe configuration register.

Transfer type	:	Specify either USB_BULK or USB_INT
BRDY operation designation	:	Specify USB_BFREOFF
BRDY operation designation	:	Specify either USB_DBLBON or USB_DBLBOFF
Double buffer mode	:	Specify either USB_CNTMDON or USB_CNTMDOFF
SHTNAK operation designation	:	Specify either USB_SHTNAKON or USB_SHTNAKOFF
Transfer direction	:	Specify either USB_DIR_H_OUT or USB_DIR_H_IN
Endpoint number	:	Specify the endpoint number (EP1 to EP15)

Restrictions

- The values that can be set for the transfer type differ according to the selected pipe. For details, refer to the hardware manual of the corresponding device.
- For a pipe set to the receive direction, specify USB_SHTNAKON

(3). Pipe definition item 3

Only for USBA/USBAa. Specify the settings for the pipe buffer designation register.

Buffer size	:	Specify the pipe buffer size in 64-byte units
Buffer start number	:	Specify the buffer start number

Restrictions

- Make settings such that buffer areas in use at the same time do not overlap
- When the USB_DBLBON setting is used, twice the area is needed

(4). Pipe definition item 4

Specify the settings for the pipe maximum packet size register

Maximum packet size	:	Specify the maximum packet size for the pipe.

Restrictions

- The Max packet size which can be specified changes with devices. For details, please refer to the hardware manual of each device

(5). Pipe definition item 5

Specify the settings for the pipe period control register.

In-buffer flush	:	Specify USBC_IFISOFF
Interval duration	:	Specify the interval value (0 to 7).

Restrictions

- When a transmission type is set up in addition to USB_ISO, the ISO IN buffer flash should set up USB_IFISOFF.
- When selecting pipes 3 to 5, specify value 0 for the interval duration.

(6). Pipe definition item 6

Specify the FIFO port to be used for the pipe.

USB_CUSE	:	CFIFO is used and CPU access is carried out.
USB_D0USE	:	D0FIFO is used and CPU access is carried out.
USB_D0DMA	:	D0FIFO is used and DTC / DMA access is carried out.
USB_D1USE	:	D1FIFO is used and CPU access is carried out.
USB_D1DMA	:	D1FIFO is used and DTC / DMA access is carried out.

Restrictions

- A transaction counter starts the pipe of the receiving direction.
- No sample functions are provided for USB_D0USE and USB_D1DMA

(7). Other pipe setting notes

- Use device class to specify transfer unit communication synchronization.
- Do not fail to write USBC_PDTBLEND at the end of the table.

4.6 Peripheral Control Transfer

This section provides a transfer control sample program which uses the API provided by the USB-BASIC-F/W. The sample illustrates a control transfer that occurs when a class request is made from PDCD.
The following functions are required for control transfer processing.

- Setup stage class request function. (This function shall match the user system. Refer to 4.6.1 Class request processing.)
- Data stage API functions (*R_usb_pstd_ControlRead* / *R_usb_pstd_ControlWrite*).
- Status stage API function (*R_usb_pstd_ControlEnd*)

Note: Class request functions must be created by the user.

4.6.1 Class request processing

When a class control request is received by USB-BASIC-F/W, the contents of the request are stored in the global variable usb_gpstd_ReqReg (type *USB_REQUEST_t*). Then, the function (the class request processing call-back) registered in *ctrltrans* of the *USB_PCDREG_t* structure is run. The USB communication structure (*USB_UTR_t)*, the request information and control transfer stage information are set as the argument of the called function. The user needs to describe the class request processing in this class request processing call-back. This class request processing function is called at the data stage and the status stage.
An example of how to create a class request processing function is described below. This is also the *ctrltrans* member of structure USB_PCDREG_t.
<**Example class request processing function** >

```
void usb_pvendor_UsrCtrlTransFunction(USB_UTR_t *ptr, USB_REQUEST_t *preq, uint16_t ctsq)
{
    if( preq->ReqTypeType == USB_CLASS )
    {
        switch( ctsq )
        {
        /* Idle or setup stage */
        case USB_CS_IDST:       usb_ pvendor _ControlTrans0(ptr); break;
        /* Control read data stage */
        case USB_CS_RDDS:       usb_ pvendor _ControlTrans1(ptr); break;
        /* Control write data stage */
        case USB_CS_WRDS:       usb_ pvendor _ControlTrans2(ptr); break;
        /* Control read nodata status stage */
        case USB_CS_WRND:       usb_ pvendor _ControlTrans3(ptr); break;
        /* Control read status stage */
        case USB_CS_RDSS:       usb_ pvendor _ControlTrans4(ptr); break;
        /* Control write status stage */
        case USB_CS_WRSS:       usb_ pvendor _ControlTrans5(ptr); break;

        /* Control sequence error */
        case USB_CS_SQER:       R_usb_pstd_ControlEnd(ptr, (uint16_t)USB_DATA_ERR);    break;
        /* Illegal */
        default:                R_usb_pstd_ControlEnd(ptr, (uint16_t)USB_DATA_ERR);    break;
        }
    }
    else
    {
        usb_cstd_SetStall(ptr, (uint16_t)USB_PIPE0);
    }
}
```

1. Data stage processing

If the received request is supported, please transfer the data to host by using the *R_usb_pstd_ControlRead*() or *R_usb_pstd_ControlWrite* API.

If the received request is not supported, return STALL to host by using R_usb_pstd_SetStall() API.

The following function is called in the class request processing function.

a). usb_ pvendor _ControlTrans1()

b). usb_ pvendor _ControlTrans2()

2. Status stage processing

If there was no problem in the setup or data stage, finish the control transfer with a proper status stage by calling R_usb_pstd_ControlEnd() with USB_DATA_END as the 2nd parameter.

If a parameter is received which is not supported in the setup stage or the data stage, return a STALL packet to host by using the usb_pstd_ControlEnd() API.

The following function is called in the class request processing function.

a). usb_ pvendor _ControlTrans3()

b). usb_ pvendor _ControlTrans4()

c). usb_ pvendor _ControlTrans5()

4.6.2 Note

Make sure that the capacity of the user buffer exceeds the transmit/receive data size (Bsize) of the control transfer data stage.

4.7 Data Transfer

4.7.1 Transfer requests

A data transfer demand is made by using R_usb_pstd_TransferStart().. For more details on the API functions described here, please refer to the corresponding API section.

4.7.2 Notification of transfer result

When a data transfer completes, the firmware notifies the PDCD of the data transfer completion and the sequence toggle bit information in the call-back function registered when the data transfer was requested.
The results of the transfer are stored in the USB_UTR_t structure member *status*.
The USB_UTR_t structure is provided to PDCD (by PCD) as the first argument of the callback function..
The following shows the transfer result.

USB_DATA_NONE	:	Data transmit normal end
USB_DATA_OK	:	Data receive normal end
USB_DATA_SHT	:	Data receive normal end with less than specified data length
USB_DATA_STALL	:	STALL response or MaxPacketSize error detected
USB_DATA_STOP	:	Data transfer forced end
USB_DATA_TMO	:	Forced end due to timeout, no call-back (uITRON only)

4.7.3 The USB Communication Structure (USB_UTR_t)

The following describes the structure members used for data transfer.
USB communication with a connected host or peripheral device, with the exception of control transfer during peripheral operation, can be accomplished by notifying the following structure to PCD.
Table4-8shows the members of the USB Transfer Structure.

Table4-8 USB Communication Structure members and description for use with peripheral

Type	Member	Description
USB_MH_t	msghead	This message header is used by the OS (or non-OS messaging system). It should not be used by the application
uint16_t	msginfo	Message type Specifies the request contents Specifies USB-BASIC-F/W using an API function Specifies USB_MSG_PCD_SUBMITUTR for USB communication
uint16_t	keyword	Sub-code Specifies the pipe number or the port number etc for USB communication
USB_REGADR_t	ipp	Set USB IP base address
uint16_t	ip	Set USB IP number
uint16_t	result	Result of USB transfer. Set by PCD/HCD.
USB_CB_t	complete	Callback function. Specifies the address of the function to be executed when USB communication ends. Use the following type declaration for the call-back function. typedef void (*USB_CB_t)(USB_UTR_t*);
void	*tranadr	USB communication buffer address Provides notification of the USB communication buffer address.
uint32_t	tranlen	USB communication data length. Specify a transfer size less or equal the user buffer size.
uint16_t	*setup	Not used in USB peripheral mode.
uint16_t	status	USB communication status PCD/HCD returns the USB communication result. Please refer to "4.7.2 Notification of transfer result ".
uint16_t	pipectr	PIPECTR register. PCD returns the PIPECTR register content. PCD specify the transfer start toggle state for continuous communication.
uint8_t	errcnt	Stores the number of errors that occurred during transfer.
uint8_t	segment	File number. Only for the ANSI-type interface.
void	*usr_data	Set the variety information.

Additional Communication Structure details for peripheral

1. Buffer address for USB communication (tranadr)
   Reception or ControlRead transfer: Specifies the address of the buffer for storing receive data.
   Transmission or ControlWrite transfer: Specifies the address of the buffer for storing transmit data.
   NoDataControl transfer: Ignored if specified.

2. USB communication data length (tranlen)
   Reception or ControlRead transfer: Stores the receive data length.
   Transmission or ControlWrite transfer: Stores the transmit data length.
   NoDataControl transfer: Set to 0.

   The remaining transmit/receive data length is stored after USB communication end. In case of control transfer in host mode, the remaining data length from the data stage is stored.

3. Pipe control (pipectr)
   PDCD can perform communication with multiple endpoint addresses over a single pipe by remembering the sequence toggle bit information in this register. When specifying transfer continuation, set the SQMON bit in this register to the previous toggle state.

4. Segment information (segment)
   Control transfer continuation: Specify USB_TRAN_CONT (continuation of transfer from data stage enabled).
   Control transfer end: Specify USB_TRAN_END.
   Data transfer continuation: Specify USB_TRAN_CONT. (Set SQMON bit in PIPECTR.)
   Data transfer end: Specify USB_TRAN_END.

4.7.4 Notes on data transfer

When the maximum packet size is an odd number and one packet is not equivalent to one transfer, the CPU may generate an address exception during buffer access.

4.7.5 Notes on data reception

Use a transaction counter for the receive pipe.
When a short packet is received, the expected remaining receive data length is stored in tranlen of USBC_UTR_t structure and this transfer is ended. When the received data exceeds the buffer size, data read from the FIFO buffer up to the buffer size and this transfer is ended. When the user buffer area is insufficient to accommodate the transfer size, the usb_cstd_DataEnd() function may clear the receive packet in some cases. (It should be used with the SHTNAK function, transaction counter function, single buffer setting, etc.
When using DTC, the buffer size must be an integral multiple of the maximum packet size.

4.7.6 Data transfer overview

The following shows an overview of data transfer with an example.
(1). Transfer is done by APL or PDCD the USB_UTR_t structure members listed below.

keyword	:	Pipe number
tranadr	:	Pointer to a data buffer containing data to be transmitted
tranlen	:	Transfer Size
segment	:	USB_TRAN_END
complete	:	Executed call-back function at the data transfer end

(2). R_usb_pstd_TransferStart() is called and data transfer request executed.
(3). After data transfer is complete, the call-back function “complete” that was set in (1) above for is called and the transfer results are notified to PDCD. Refer to “4.7.2 Notification of transfer result”.

4.8 Peripheral Battery Charging (PBC)

PBC is the H/W control program for the target device that operates the Charging Port Detection (CPD) defined by the USB Battery Charging Specification (Revision 1.2).
USB-BASIC-F/W notifies the result of the CPD action to the upper layer by the callback function which is set in the member *devdefault* of USB_PCDREG_t structure, using the second argument. Please refer to “4.4.1 USB_PCDREG_t structure” about USB_PCDREG_t structure.

The result of the callback notified to the upper layer is one of the following:

0 : Standard Downstream Port (SDP) Detection
1 : Charging Downstream Port (CDP) Detection
2 : Dedicated Charging Port (DCP) Detection

The processing flow of PBC is shown in Figure 4-1.

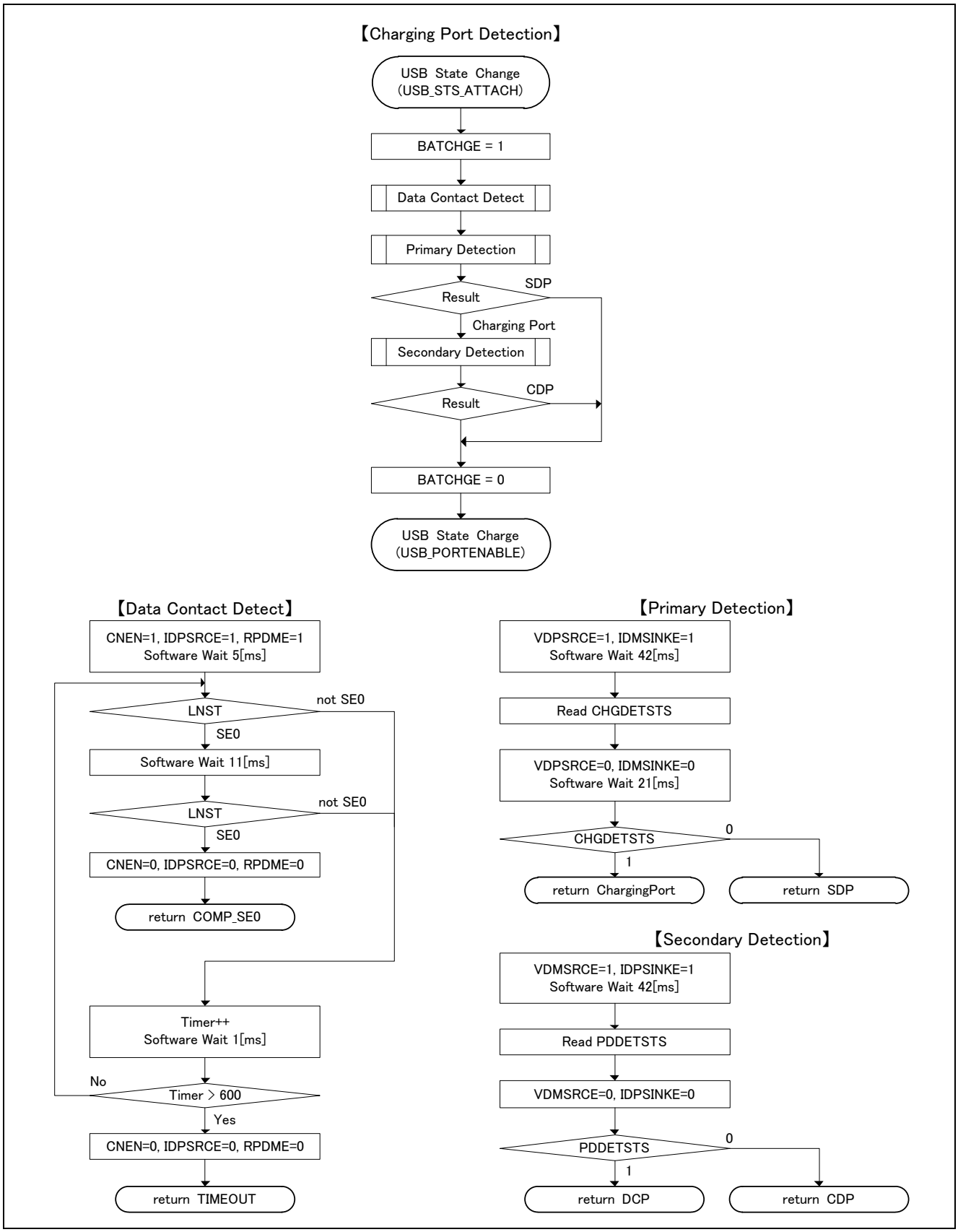

Figure 4-1 PBC processing flow

4.9 How to operate USB-BASIC-F/W FIT module in peripheral mode

This section describes the procedure for operating the USB-BASIC-F/W FIT module in Peripheral mode.

4.9.1 User Configuration File - r_usb_basic_config.h

Functional settings for the USB-BASIC-F/W module are specified by overwriting the user-defined information file (r_usb_basic_config.h). Modify the items below to match the specifications of the system.

(1). DTC FIT module setting

Selects whether or not the DTC FIT module is used.

#define USB_DTC_ENABLE : DTC FIT module used
#define USB_DTC_ENABLE : DTC FIT module not used

(2). USB operating mode (Host/Peripheral) setting

Specifies the operating mode (Peripheral mode) per the USB module (USBb, USBA/USBAa). If USB module is not used, specifies USB_NOUSE_PP.

#define USB_FUNCSEL_USBIP0_PP	USB_PERI_PP	: Peripheral mode for USBb
#define USB_FUNCSEL_USBIP0_PP	USB_NOUSE_PP	: USBb not used
#define USB_FUNCSEL_USBIP1_PP	USB_PERI_PP	: Peripheral mode for USBA/USBAa
#define USB_FUNCSEL_USBIP1_PP	USB_NOUSE_PP	: USBA/USBAa not used

[Note]

Don't define USB_PERI_PP to USB_FUNCSEL_USBIP0_PP and USB_FUNCSEL_USBIP1_PP simultaneously.

(3). USB module type setting

Specifies the USB module type of USB module (USBA/USBAa).

#define USB_SPEED_PP USB_HS_PP :Hi-speed module type setting of USBA/USBAa module
#define USB_SPEED_PP USB_FS_PP :Full-speed module setting of USBA/USBAa module

[Note]

Be sure to set USB_HS_PP when using RX64M.

(4). Battery Charging functionality in Peripheral mode

Specifies whether or not battery charging functionality is enabled during peripheral operation. To use battery charging functionality, enable this macro.
This item is valid when the setting of USB_FUNCSEL_USBIP0_PP/USB_FUNCSEL_USBIP1_PP is USB_PERI_PP.

#define USB_PERI_BC_ENABLE : Battery Charging function used
#define USB_PERI_BC_ENABLE : Battery Charging function not used

(5). FIFO access endian setting

Specify the endian mode for FIFO access.

#define USB_CPUBYTE_PP	USB_BYTE_LITTLE_PP	: Little endian
#define USB_CPUBYTE_PP	USB_BYTE_BIG_PP	: Big endian

(6). Low-power mode setting

Specify whether or not low-power mode is used.

#define USB_CPU_LPW_PP	USB_LPWR_USE_PP	: Low Power Mode used
#define USB_CPU_LPW_PP	USB_LPWR_NOT_USE_PP	: Low Power Mode not used

(7). Task etc ID maximum value and Task priority value

Set the value to the following definition depending on User system. RAM size changes by this value.

a) USB_IDMAX
b) USB_PRIMAX
c) USB_BLKMAX

5. Host

5.1 Host Control Driver (HCD)

5.1.1 Basic function

HCD is a program for controlling the hardware. The functions of HCD are shown below.

1. Control transfer (Control Read, Control Write, No-data Control) and result notification.
2. Data transfer (bulk, interrupt) and result notification.
3. Data transfer suspension (all pipes).
4. USB communication error detection and automatic transfer retry
5. USB bus reset signal transmission and reset handshake result notification.
6. Suspend signal and resume signal transmission.
7. Attach/detach detection using ATCH and DTCH interrupts.
8. Hardware control when entering and returning from the clock stopped (low-power sleep mode) state.(Only R8A66597)

5.1.2 Issuing Requests to HCD

The API functions described below are used by a higher level task (the HDCD or APL) to issue hardware control requests to HCD. Only HCD may directly control the hardware. For requests to update the state of the connected devices (Powered, Address, Configured state etc), control may be exercised by HCD either directly or via a USB hub. MGR determines the device address and issues control requests to the HCD and HUBCD tasks.
In response to a request from upper layer task, HCD sends a result notification by means of a call-back function.
The following points must be considered when using HCD to perform USB communication.

1. Bus possession rate and pipe contention

HCD can perform communication with multiple devices. However, HCD does not calculate the USB bus possession rate or check for contention between the communication pipes used by HDCD tasks.
When multiple HDCD devices are installed, modifications must be made to prevent contention between pipes.
Also note that the HUBCD provided in the USB-BASIC-F/W uses pipe 9, which may affect operations when using HUBCD in the user system.

2. Installation of multiple HDCD (class drivers)

Two or more devices and communication are possible using HCD, but MGR cannot proceed with enumeration of multiple devices simultaneously. The application will have to keep track of the communication if several devices are running. Enumeration of another device should be avoided until the enumeration of the previously connected device finishes.

5.2 Host Manager (MGR)

5.2.1 Basic function

MGR is a task that supplements the functions of HCD and HDCD. The functions of MGR are shown below.

1. Registration of HDCD.
2. State management for connected devices.
3. Enumeration of connected devices.
4. Searching for endpoint information from descriptors.

5.2.2 USB Standard Requests

MGR enumerates connected devices. The USB standard requests issued by MGR are listed below. The descriptor information obtained from a device is stored temporarily, and this information can be fetched by using the HCD API function.

GET_DESCRIPTOR（Device Descriptor）
SET_ADDRESS
GET_DESCRIPTOR（Configuration Descriptor）
SET_CONFIGURATION

5.2.3 Checking of HDCD

MGR notifies HDCD of the information obtained during enumeration by using the GET_DESCRIPTOR request and checks whether the connected device is ready to operate.
HDCD replies device condition by R_usb_hstd_ReturnEnuMGR() function.

5.3 API Information

This Driver API follows the Renesas API naming standards.

5.3.1 Hardware Requirements

This driver requires your MCU support the following features:

- USB

5.3.2 Software Requirements

This driver is dependent upon the following packages:

- r_bsp
- r_usb_dtc (using DTC transfer)

5.3.3 Supported Toolchains

This driver is tested and working with the following toolchains:

- Renesas RX Toolchain v.2.01.00b

5.3.4 Header Files

All API calls and their supporting interface definitions are located in r_usb_basic_if.h.

5.3.5 Integer Types

This project uses ANSI C99 “Exact width integer types” in order to make the code clearer and more portable. These types are defined in *stdint.h*.

5.3.6 Compile Setting

Please modify r_usb_basic_config.h when User sets the module configuration option.
The following table shows the option name and the setting value.

Configuration options in r_usb_basic_config.h	
USB_DTC_ENABLE	Selects whether or not the DTC FIT module is used. = Enabled: DTC FIT module is used. = Disabled: DTC FIT module is not used.
USB_FUNCSEL_USBIP0_PP	Specifies the operating mode of the USB module (USBb). = USB_HOST_PP: USB module operates as a host. = USB_NOUSE_PP: USB module does not operate.
USB_FUNCSEL_USBIP1_PP	Specifies the operating mode of the USB module (USBA/USBAa). = USB_HOST_PP: USB module operates as a host. = USB_NOUSE_PP: USB module does not operate.
USB_SPEED_PP	Specifies the speed mode of the USB module (USBA/USBAa). = USB_HS_PP: Use Hi-speed (RX71M only) = USB_FS_PP: Use Full-speed
USB_HOST_BC_ENABLE	Specifies whether or not Battery Charging functionality is enabled in Host mode. = Enabled: Battery charging functionality is used. = Disabled: Battery charging functionality is not used.
USB_BC_DCP_ENABLE	Specifies whether or not the USB module functions as a dedicated charging port when Battery Charging functionality is enabled in Host mode. = Enabled: Used as dedicated charging port (DCP). = Disabled: Used as charging downstream port (CDP).
USB_BC_ATTACH	Registers the function that is called when a device is attached in Host mode. This item is valid when battery charging functionality is used.
USB_CPUBYTE_PP	Specifies the endian mode for USB FIFO register access. The setting of this item should match the endian mode of the CPU. = USB_BYTE_LITTLE_PP: Little-endian = USB_BYTE_BIG_PP: Big-endian
USB_CPU_LPW_PP	Specifies whether or not low-power mode is used. = USB_LPWR_USE_PP: Low-power mode is used.

	= USB_LPWR_NOT_USE_PP: Low-power mode is not used.
USB_OVERCURRENT	Registers the function that is called when overcurrent detection occurs in Host mode.
USB_HOST_COMPLIANCE_MODE	Specifies whether or not USB embedded host compliance testing is supported in Host mode. = Enabled: Compliance testing is supported. = Disabled: Compliance testing is not supported.
USB_ COMPLIANCE_DISP	Registers the function for displaying information on the display device (LCD, etc.) in enabling USB_HOST_COMPLIANCE_MODE.
USB_HS_EL_TEST	Enables this definition when doing Hi-speed electrical test in enabling USB_HOST_COMPLIANCE_MODE
USB_RESPONCE_COUNTER_VALUE	Specifies in ms units the time that must elapse before the device is determined to have returned no response to a control transfer in enabling USB_HOST_COMPLIANCE_MODE.

5.3.7 Argument

The structure for the arguments of the API functions is shown below. For details, please refer to USB Basic Firmware application note (R01AN2025EJ).

```
typedef struct USB_UTR
{
  USB_MH_t      msghead;              /* Message header (for RTOS) / No used */
  uint16_t      msginfo;              /* Message information */
  uint16_t      keyword;              /* Sub information(Port No/Pipe No etc)  */
  union {
        USB_REGADR_t      ipp;      /* USB IP address (for USB0)*/
        USB_REGADR1_t     ipp1;     /* USB IP address (for USB1/USBA) */
  };
  uint16_t      ip;                   /* USB IP number*/
  uint16_t      result;               /* USB transfer result */
  USB_CB_t      complete;             /* Pointer to the callback function */
  void          *tranadr;             /* Pointer to USB transfer buffer */
  uint32_t      tranlen;              /* USB data length*/
  uint16_t      *setup;               /* Pointer to Setup packet data */
  uint16_t      status;               /* USB status */
  uint16_t      pipectr;              /* PIPECTR register */
  uint8_t       errcnt;               /* Error count for transferring */
  uint8_t       segment;              /* Segment information */
  void          *usr_data;            /* Other various information */
} USB_UTR_t;
```

5.3.8 Adding the Module

This module must be added to an existing e^2 studio project. By using the e^2 studio plug-in, it is possible to update the include file path automatically. It is therefore recommended that this plug-in be used to add the project.

For instructions when using e^2 studio, refer to RX Family: Integration into e^2 studio, Firmware Integration Technology (document No. R01AN1723EU).

5.4 API (Application Programming Interface)

A HDCD module will implement hardware control requests by using the following HCD API. The return values of each API function are scheduler macro error codes. A list of HCD API functions is shown in Table5-1.

Table5-1 HCD API function list

Function	Description
R_USB_Open()	Start the USB module
R_USB_Close()	Stop the USB module
R_USB_GetVersion()	Return the driver version
R_usb_hstd_TransferStart()	Data transfer request
R_usb_hstd_TransferEnd()	Data transfer forced end request
R_usb_hstd_SetPipeRegistration()	Pipe configuration setting request
R_usb_hstd_ChangeDeviceState()	Connected device state change request
R_usb_hstd_HcdOpen()	Start HCD Task
R_usb_hstd_HcdClose()	End HCD Task
R_usb_hstd_HcdTask()	HCD Task
R_usb_hstd_DriverRegistration()	Register HDCD
R_usb_hstd_SetPipeInfo()	Set of pipes information
R_usb_hstd_DeviceInformation()	Fetch the state of a connected peripheral
R_usb_hstd_ChkPipeInfo	Create pipe information from endpoint descriptor
R_usb_hstd_ReturnEnuMGR()	Enumeration continuation request (non-OS only)
R_usb_hstd_EnuWait()	Enumeration priority update request (non-OS only)
R_usb_hstd_MgrOpen()	Start MGR Task
R_usb_hstd_MgrClose()	End MGR Task
R_usb_hstd_MgrTask()	MGR Task
R_usb_cstd_GetUsbIpAdr()	Get USB register base address
R_usb_cstd_UsbIpInit()	Initialize USB module
R_usb_cstd_ClearHwFunction()	USB-Related Register Initialization Request
R_usb_cstd_SetRegDvstctr0()	Set the value to DVSTCTR0 register

5.4.1 R_USB_Open

Power on USB module.

Format

usb_err_t R_USB_Open(usb_ip_t ip_type)

Arguments

ip_type IP number (USB_IP0/USB_IP1) of USB module.

Return Value

USB_SUCCESS	Success
USB_ERR_BUSY	Used USB module by other S/W module.
USB_ERR_OPENED	USB module currently in operation

Description

This function applies power to the USB module specified in the argument. USBb module is powered on when specifing “USB_IP0” in the argument, USBA/USBAa module is powered on when specifing “USB_IP1” in the argument.

Note

1. Don’t call this API after powering on USB module.

Example

```
void  usb_smp_task()
{
  USB_ER_t    err;
        :
  err = R_USB_Open(USB_IP0);        /* Start USBb module */
  if(err != USB_SUCCESS)
  {
    /* error */
  }
        :
}
```

5.4.2 R_USB_Close

Power off USB module.

Format

usb_err_t R_USB_Close(usb_ip_t ip_type)

Arguments

ip_type IP number (USB_IP0/USB_IP1) of USB module.

Return Value

USB_SUCCESS	Success
USB_ERR_BUSY	Locked USB module by other S/W module
USB_ERR_NOT_OPEN	USB module is not opened.

Description

This function removes power to the USB module specified in the argument. USBb module is powered off when specifing “USB_IP0” in the argument, USBA/USBAa module is powered off when specifing “USB_IP1” in the argument.

Note

--

Example

```
void  usb_smp_task()
{
  USB_ER_t    err;
        :
  err = R_USB_Close(USB_IP0);         /* Stop USBb module */
  if(err != USB_SUCCESS)
  {
    /* error */
  }
        :
}
```

5.4.3 R_USB_GetVersion

Return the driver version number

Format

uint32_t R_USB_GetVersion(void)

Arguments

— —

Return Value

Version number

Description

This function returns the driver version number at runtime

Note

--

Example

```
void  usb_smp_task()
{
  uint32_t   version;
       :
  version = R_USB_GetVersion();
       :
}
```

5.4.4 R_usb_hstd_TransferStart

Data transfer request

Format

USB_ER_t R_usb_hstd_TransferStart(USB_UTR_t *ptr)

Argument

*ptr Pointer to USB transfer structure

Return Value

USB_E_OK Success

USB_E_ERROR Failure

Description

This function requests HCD to execute data transfer(s) for the pipe specified in the Transfer Structure.

When the data transfer ends (specified data size reached, short packet received, error occurred), the call-back function is called. The information of the remaining transmit/receive data length, status, error count and transfer end is set in the argument to this call-back function.

Note

1. Set the following members of the USB Transfer Structure before calling this function; ip, ipp, pipe number, transfer data start address, transfer data length, status, and the call-back function to call at end of transfer.
2. Call this function from the user application or class driver (HDCD).
3. When the received data is n times of the maximum packet size and less than the expected received data length, it is considered that the data transfer is not ended and a callback is not generated
4. If the USB device in other USB port is in the USB enumeration, this API returns USB_E_QOVR by completing the enumeration.

Example

```
USB_UTR_t   usb_smp_trn_Msg[USB_NUM_USBIP][USB_MAXPIPE_NUM + 1];
USB_ER_t usb_smp_task(USB_UTR_t *ptr, uint16_t pipe, uint32_t size, uint8_t *table)
{
    :
  /* Transfer information setting */
  trn_msg[ptr->ip][pipe].msghead  = (USB_MH_t)NULL;   /* NULL only */
  trn_msg[ptr->ip][pipe].keyword  = pipe;  /* Pipe no. */
  trn_msg[ptr->ip][pipe].tranadr  = table; /* Pointer to data buffer */
  trn_msg[ptr->ip][pipe].tranlen  = size;  /* Transfer size */
  trn_msg[ptr->ip][pipe].setup    = 0;
  trn_msg[ptr->ip][pipe].complete = ptr->complete;  /* Call-back function */
  trn_msg[ptr->ip][pipe].msghead  = USB_TRAN_END;  /* Status */
  trn_msg[ptr->ip][pipe].ipp      = ptr->ipp;  /* USB IP base address */
  trn_msg[ptr->ip][pipe].ip       = ptr->ip;   /* USB IP No */

  /* Data transfer request */
  err = R_usb_hstd_TransferStart((USB_UTR_t *)&trn_msg [ptr->ip][pipe]);

  return err;
    :
}
```

5.4.5 R_usb_hstd_TransferEnd

Data transfer forced end request

Format

USB_ER_t R_usb_hstd_TransferEnd(USB_UTR_t *ptr, uint16_t pipe, uint16_t status)

Argument

*ptr	Pointer to USB transfer structure
pipe	Pipe number
status	USB communication status

Return Value

USB_E_OK	Success
USB_E_ERROR	Failure

Description

This function forces data transfer via the pipes to end.

This “forced end” request is sent to HCD which does the forced end request processing.

When a data transfer is forcibly ended, the function calls the call-back function that was set by *R_usb_pstd_TransferStart* when the data transfer was requested. The remaining data length of transmission and reception, status, the number of times of a transmission error, and the information on forced termination are set in the argument (ptr) of this callback function.

Note

1. Besides above arguments, also set the following members of the USB Transfer Structure before calling the function.
   USB_REGADR_t ipp :USB register base address
   uint16_t ip :USB IP number
2. Call this function from the user application or the class driver (HDCD).

Example

```
void  usb_smp_task(USB_UTR_t *ptr)
{
  uint16_t status;
  uint16_t pipe;
    :
  pipe    = USB_PIPEx
  status  = USB_DATA_STOP

  /* Transfer end request */
  err = R_usb_hstd_TransferEnd(ptr, pipe, status);

  return err;
    :
}
```

5.4.6 R_usb_hstd_SetPipeRegistration

Pipe configuration setting request

Format

USB_ER_t	R_usb_hstd_SetPipeRegistration(USB_UTR_t *ptr, uint16_t *table, uint16_t pipe)

Argument

*ptr	Pointer to USB transfer structure
*table	Pointer to pipe information table
pipe	Pipe number

Return Value

–	Error code. (USB_E_OK)

Description

This function configures the hardware pipes. Each pipe is set according to the contents of the pipe information registered during HDCD registration.

Note

1. Besides above arguments, also set the following members of the USB Transfer Structure before calling the function.
   USB_REGADR_t ipp :USB register base address
   uint16_t ip :USB IP number
2. The user must call this function from the user's application during initialization.

Example

```
USB_ER_t usb_smp_task(USB_UTR_t *ptr)
{
    :
  R_usb_hstd_SetPipeRegistration(ptr, (uint16_t*)&usb_ smp_eptbl, USB_USEPIPE);
    :
}
```

5.4.7

R_usb_hstd_HcdOpen

Start HCD Task

Format

USB_ER_t R_usb_hstd_HcdOpen(USB_UTR_t *ptr)

Argument

*ptr Pointer to a USB Transfer Structure

Return Value

USB_E_OK Success

Description

This function initializes (clears) the host's pipe information.

Return value is USB_E_OK at any time.

Note

1. Besides above arguments, also set the following members of the USB Transfer Structure before calling the function.
   USB_REGADR_t ipp :USB register base address
   uint16_t ip :USB IP number
2. The user program should register the HDCD in the HCD and then call this function during initialization.
3. Do not call this function after starting the HCD task.

Example

```
USB_ER_t usb_smp_task(USB_UTR_t *ptr)
{
  USB_UTR_t  *ptr;
    :
  R_usb_hstd_HcdOpen(ptr);
    :
}
```

5.4.8 R_usb_hstd_HcdClose

End HCD Task

Format

USB_ER_t	R_usb_hstd_HcdClose(void)

Argument

—	—

Return Value

USB_E_OK	Success

Description

No processing.
Return value is always USB_E_OK at any time.

Note

1. Call this function from the user application or class driver (HDCD).
2. Please do not call this function after ending HCD task.

Example

```
void  usb_smp_task()
{
  USB_UTR_t  *ptr;
    :
  R_usb_hstd_HcdClose(ptr);
    :
}
```

5.4.9 R_usb_hstd_HcdTask

HCD Task

Format

void R_usb_hstd_HcdTask(USB_VP_INT stacd)

Argument

stacd Task start code (Not used)

Return Value

— —

Description

When host, call this function regularly so that the USB H/W IP is controlled continuously by HCD.

Note

1. Besides above arguments, also set the following members of the USB Transfer Structure before calling the function.
   USB_REGADR_t ipp :USB register base address
   uint16_t ip :USB IP number
2. Call this function in the loop that executes the scheduler processing for non-OS operations.

Example

```
void  Start non-OS scheduler (void)
{
  while(1)
  {
  /* Start non-OS scheduler */
  R_usb_cstd_Scheduler();
  /* Flag checking */
  if(USB_FLGSET == R_usb_cstd_CheckSchedule())
  {
       :
    R_usb_hstd_HcdTask((USB_VP_INT)0);
       :
  }
    :
}
```

5.4.10 R_usb_hstd_DriverRegistration

Register HDCD

Format

void	R_usb_hstd_DriverRegistration(USB_UTR_t *ptr, USB_HCDREG_t *callback)

Argument

*ptr	Pointer to USB transfer structure
*callback	Pointer to class driver structure

Return Value

—	—

Description

This function registers the HDCD information, which is registered in the class driver structure, in the HCD.

It then updates the number of registered drivers controlled by HCD, and registers HDCD in a new area.

After registration is complete, the initialization call-back function is executed.

Note

1. Besides above arguments, also set the following members of the USB Transfer Structure before calling the function.
   USB_REGADR_t ipp :USB register base address
   uint16_t ip :USB IP number
2. The user must call this function from the user application during initialization.
3. Refer to ‘Table5-7 USB_HCDREG_t structure **members**’ for details about registration information.

Example

```
void  usb_smp_registration(USB_UTR_t *ptr)
{
  USB_HCDREG_t  driver;
    :
   (Register information of HDCD to driver)
    :
   R_usb_hstd_DriverRegistration(ptr, &driver);
}
```

5.4.11 R_usb_hstd_SetPipeInfo

Copy pipe information to host pipe information table

Format

void	R_usb_hstd_SetPipeInfo(uint16_t *dst_pipe_tbl, uint16_t *src_pipe_tbl, uint16_t length)

Argument

*dst_pipe_tbl	Pointer to pipe information table (destination)
* src_pipe_tbl	Pointer to pipe information table (source)
length	Length of table

Return Value

—	—

Description

Pipe information table is copied from source (*tmp_tble) to destination (*ep_tbl).

Note

Call this function from the user application or class driver.

Example

```
void  usb_smp_task(void)
{
    :
  R_usb_hstd_SetPipeInfo(&usb_def_eptbl[0], &usb_tmp_eptbl[offset], length);
    :
}
```

5.4.12 R_usb_hstd_DeviceInformation

Request device information for connected peripheral

Format

void R_usb_hstd_DeviceInformation(USB_UTR_t *ptr, uint16_t devaddr, uint16_t *tbl)

Argument

*ptr	Pointer to USB transfer structure
devaddr	Device address
*tbl	Pointer to the table address where acquired device information is stored

Return Value

— —

Description

Information about the peripheral connected to the USB port is acquired.

The following information is stored in a device information table is shown below.

［0］ Root port number to which device is connected
［1］ Device state
- USB_DETACHED (10) : Un connect
- USB_DEFAULT (40) : Enumeration state
- USB_CONFIGURED (70) : Configured state
- USB_SUSPENDED (80) : Suspend state
- USB_SUSPENDED_PROCESS (102) : Moving to Suspend state

［2］ Configuration number
［3］ Interface class code 1
［4］ Connection speed
- USB_HSCONNECT (0xC0) : High Speed
- USB_FSCONNECT (0x80) : Full Speed
- USB_LSCONNECT (0x40) : Low Speed
- USB_NOCONNECT (0x00) : Powerd state or non connect

［5］ Number of interfaces used
［6］ Interface class code 2
［7］ Interface class code 3
［8］ Status of rootport 0
［9］ Status of rootport 1

Note

1. Besides above arguments, also set the following members of the USB Transfer Structure before calling the function.
   USB_REGADR_t ipp :USB register base address
   uint16_t ip :USB IP number
2. Call this function from the user application or class driver.
3. If this API is called in the state of other than Configured state, set 0(zero) to the argument “devaddr”. When specifying 0 to the device address, the following information is returned.
   (1). When there is not a device during enumeration.
   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
4. As USB-BASIC-F/W does not support multiple interfaces, [5], [6] and [7] above are not used.
5. Use a 20-byte area for argument *tbl.

Example

```
void  usb_smp_task(void)
{
    :
  /* Check device information */
  R_usb_hstd_DeviceInformation(ptr, devaddr, &tbl);
    :
}
```

5.4.13 R_usb_hstd_ChkPipeInfo

Setting the pipe information table

Format

uint16_t R_usb_hstd_ChkPipeInfo(uint16_t speed, uint16_t *EpTbl, uint8_t *Descriptor);

Argument

speed	Device Speed
EpTbl	Pipe Information Table
Descriptor	Endpoint Descriptor

Return Value

USB_DIR_H_IN	Set the IN endpoint.
USB_DIR_H_OUT	Set the OPUT endpoint.
USB_E_ERROR	Failure

Description

Analyze the endpoint descriptor and create the pipe information table for a pipe.
Fields where information is updated are as follows:

USB_TYPFIELD	USB_BULK .or. USB_INT
USB_SHTNAKFIELD	USB_SHTNAKON（USB_TYPFIELD == USB_DIR_H_IN）
USB_DIRFIELD	USB_DIR_H_IN .or. USB_DIR_H_OUT
USB_EPNUMFIELD	Endpoint number shown in the endpoint descriptor
USB_IITVFIELD	Interval counter (specified by 2 to the nth power)

Note

1. Call this function from the user application or class driver.
2. Set the interval counter by 2 to the nth.

Example

```
void usb_hsmpl_pipe_info(uint8_t *table)
{
 usb_er_t   retval = USB_YES;
 uint16_t   *ptr;

 /* Check Endpoint Descriptor */
 ptr  = g_usb_hsmpl_DefEpTbl;
 for (; table[1] == USB_DT_ENDPOINT, retval != USB_ERROR; table += table[0],
ptr += USB_EPL)
 {
  retval = R_usb_hstd_ChkPipeInfo(speed, &ptr, table);
 }
 return retval;
}
```

5.4.14 R_usb_hstd_ReturnEnuMGR

Enumeration continuation request (non-OS only)

Format

void R_usb_hstd_ReturnEnuMGR(USB_UTR_t *ptr, uint16_t cls_result)

Argument

*ptr	Pointer to USB transfer structure
cls_result	Class check result

Return Value

—	—

Description

The function transmits a message to MGR requesting continuation of enumeration processing. MGR analyzes the argument of the function and determines whether or not to transition the connected peripheral device to the configured state. Specify the class check result as the argument (cls_result).

Note

1. Besides above arguments, also set the following members of the USB Transfer Structure before calling the function.
   USB_REGADR_t ipp :USB register base address
   uint16_t ip :USB IP number
2. This function is for non-OS operations only.
3. Please call at the time of class check processing.

Example

```
void  usb_smp_task(void)
{
    :
   (Enumeration processing)
    :
  R_usb_hstd_ReturnEnuMGR(ptr, cls_result);
    :
}
```

5.4.15 R_usb_hstd_EnuWait

Enumeration priority update request (non-OS only)

Format

void	R_usb_hstd_EnuWait(USB_UTR_t *ptr, uint8_t taskID)

Argument

*ptr	Pointer to USB transfer structure
taskID	Task ID

Return Value

—	—

Description

An Enumeration priority change request is made. It can be used to change the enumeration priority in cases where a single piece of hardware has multiple USB ports and, during enumeration of one USB device, attachment of another USB device is detected.

Note

1. Besides above arguments, also set the following members of the USB Transfer Structure before calling the function.
   USB_REGADR_t ipp :USB register base address
   uint16_t ip :USB IP number
2. This function is for non-OS operations only.
3. Please call at the time of class check processing.

Example

```
void  usb_smp_task(void)
{
    :
  R_usb_hstd_EnuWait(ptr, (uint8_t)USB_HSMP_TSK);
    :
}
```

5.4.16 R_usb_hstd_ChangeDeviceState

Connected device state change request

Format

USB_ER_t R_usb_hstd_ChangeDeviceState(USB_UTR_t *ptr, USB_CB_t complete, uint16_t msginfo, uint16_t devaddr)

Argument

*ptr	Pointer to a USB Transfer Structure
complete	Call-back function
msginfo	Message information
devaddr	Device address

Return Value

USB_E_OK	Success
USB_E_ERROR	Failure

Description

Setting the following values in msginfo and calling this API will send a request to HCD or HUBCD to change the state of the connected USB device.

msginfo	Outline
USB_DO_RESET_AND_ENUMERATION	Request for transition from connected state to communication initialized connected state. (issue USB reset and request enumeration execution)
USB_PORT_ENABLE	VBUS supply start request
USB_PORT_DISABLE	VBUS shutdown request
USB_DO_GLOBAL_SUSPEND	Request for transition to suspended state with remote wakeup enabled
USB_DO_GLOBAL_RESUME	Global resume execution request (request transition from suspend state to state preceding suspend)
USB_DO_SELECTIVE_RESUME	Selective resume execution request (request transition from suspend state to state preceding suspend)
USB_DO_CLR_STALL	Clear stall request
USB_DO_SET_SQTGL	Sequence toggle bit (SQSET) set request
USB_DO_CLR_SQTGL	Sequence toggle bit (SQCLR) clear request

Note

1. Besides above arguments, also set the following members of the USB Transfer Structure before calling the function.
   USB_REGADR_t ipp :USB register base address
   uint16_t ip :USB IP number
2. Call this function from the user application or class driver.

Example

```
void  usb_smp_task(void)
{
    :
  R_usb_hstd_ChangeDeviceState(ptr,(USB_CB_t)&usb_smp_callback,
                                    msginfo, devaddr);
    :
}
```

5.4.17 R_usb_hstd_MgrOpen

Start the MGR Task

Format

USB_ER_t R_usb_hstd_MgrOpen(USB_UTR_t *ptr)

Argument

*ptr Pointer to a USB Transfer Structure

Return Value

USB_E_OK Success

Description

The registration state of HDCD is initialized.
Return value is USB_E_OK at any time.

Note

1. Besides above arguments, also set the following members of the USB Transfer Structure before calling the function.
   USB_REGADR_t ipp :USB register base address
   uint16_t ip :USB IP number
2. The user application should register the PDCD in the PCD and then call this function during initialization.
3. Do not call this function after starting the MGR task.

Example

```
void  usb_smp_task()
{
  USB_UTR_t  *ptr;
    :
  R_usb_hstd_MgrOpen(ptr);
    :
}
```

5.4.18 R_usb_hstd_MgrClose

End MGR Task

Format

USB_ER_t	R_usb_hstd_MgrClose(void)

Argument

*ptr	Pointer to USB transfer structure

Return Value

—	—

Description

No processing.
Return value is always USB_E_OK.

Note

1. Please call this function from the user application or class driver.
2. Please do not call this function after ending MGR task.

Example

```
void  usb_smp_task()
{
    :
  R_usb_hstd_MgrClose();
    :
}
```

5.4.19 R_usb_hstd_MgrTask

MGR Task

Format

void	R_usb_hstd_MgrTask(USB_VP_INT_t stcd)

Argument

stacd	Task start code (Not used)

Return Value

—	—

Description

When host, call this function continuously to run the Manager (MGR) task. (HDCD sends messages to MGR which in turn calls HCD.

Note

1. Call this in the loop that executes the scheduler processing.

Example

```
void  usb_smp_mainloop(void)
{
  while(1)
  {
  /* Start scheduler */
  R_usb_cstd_Scheduler();
  /* Flag checking */
  if(USB_FLGSET == R_usb_cstd_CheckSchedule())
  {
        :
   R_usb_hstd_MgrTask((USB_VP_INT)0);
        :
  }
        :
}
```

5.4.20 R_usb_cstd_GetUsbIpAdr

Get USB register base address

Format

void R_usb_cstd_GetUsbIpAdr(uint16_t usbip)

Argument

usbip USB IP number

Return Value

USB register base address

Description

Return USB register base address of specified USB IP .

Note

1. Call this function from the user application.

Example

```
void  usb_smp_task( void )
{
    USB_UTR_t  utr;
               :
    utr.ip = USB_HOST_USBIP_NUM;
    utr.ipp = R_usb_cstd_GetUsbIpAdr(USB_HOST_USBIP_NUM );
               :
}
```

5.4.21 R_usb_cstd_UsbIpInit

Initialize USB module

Format

void R_usb_cstd_UsbIpInit(USB_UTR_t *ptr, uint16_t mode)

Argument

*ptr	Pointer to a USB Transfer Structure
mode	USB operation mode (USB_HOST, USB_PERI)

Return Value

—

Description

This API initializes USB module beased on the second argument.

USB_HOST	: Host mode setting
USB_PERI	: Peripheral mode setting

Note

1. Besides above arguments, also set the following members of the USB Transfer Structure before calling the function.

USB_REGADR_t	ipp	:USB register base address
uint16_t	ip	:USB IP number

2. The user must call this function from the user application during initialization.

Example

```
void  usb_smp_init( void )
{
        :
    USB_UTR_t  utr;
       :
    utr.ip = USB_HOST_USBIP_NUM;
    utr.ipp = R_usb_cstd_GetUsbIpAdr(USB_HOST_USBIP_NUM );
       :
    R_usb_cstd_UsbIpInit( &utr );
       :

}
```

5.4.22 R_usb_cstd_ClearHwFunction

USB-Related Register Initialization

Format

void R_usb_cstd_ClearHwFunction(USB_UTR_t *ptr)

Argument

*ptr Pointer to a USB Transfer Structure

Return Value

—

Description

USB-Related Register Initialization Request

Note

1. Besides above arguments, also set the following members of the USB Transfer Structure before calling the function.

USB_REGADR_t	ipp	:USB register base address
uint16_t	ip	:USB IP number

5.4.23 R_usb_cstd_SetRegDvstctr0

Set DVSTCTR0 register

Format

void	R_usb_cstd_SetRegDvstctr0(USB_UTR_t *ptr, uint16_t data)

Argument

*ptr	Pointer to a USB Transfer Structure
data	Setting value to register

Return Value

—	—

Description

Set the value of the second argument to the DVSTCTR0 register.
DVSTCTR0 controls the state of the USB data bus.

Note

1. Besides above arguments, also set the following members of the USB Transfer Structure before calling the function.
   USB_REGADR_t ipp :USB register base address
   uint16_t ip :USB IP number
2. Please call this function from the class driver (HDCD).

5.5 Host call-back functions

5.5.1 HCD call-backs

Call-backs are used so that the user does not need to poll for events. The user application registers a callback in order to be alerted when, for example, requests made to HCD are completed, and data is has been transferred over USB to the peripheral. By using call-backs the application is propelled forward by the USB USB-BASIC-F/W stack.
Each table below is named after the function *registering* the call-back, and the table content explains the nature of the "called-back" function.

Table5-2 R_usb_hstd_TransferStart call-back

Name	Data Transfer Request call-back function		
Call format	(*USB_CB_INFO_t)(USB_UTR_t*);		
Arguments	USB_UTR_t*		USB_UTR_t pointer of argument of R_usb_hstd_TransferStart
Return values	—	—	—
Description	This function is executed at the end of a data transfer. This should occur when the number of bytes specified by the TransferStart function are transmitted, or when a short packet is received. Transmit/receive data length and the error count are updated. Check the result of data communications.		
Notes			

Table5-3 R_usb_hstd_ChangeDeviceState(USB_MSG_HCD_USBRESET) call-back

Name	USB Bus Reset call-back function		
Call format	(*USB_CB_INFO_t)(USB_UTR_t*,uint16_t,uint16_t)		
Arguments	USB_UTR_t*		Pointer to a USB Transfer Structure
	uint16_t		USB_NOCONNECT: Not connected USB_HSCONNECT: Hi-speed device USB_FSCONNECT: Full-speed device USB_LSCONNECT: Low-speed device
	uint16_t		NOARGUMENT: Not used
Return values	—	—	—
Description	This function is executed at the end of USB bus reset processing. The communication speed of the connected device is returned as an argument to the call-back. The response is "not connected" if disconnection is detected during USB bus reset processing or if the speed is undefined.		
Notes			

Table5-4 Other call-backs

Name	Other Call back Functions		
Call format	(*USB_CB_INFO_t)(USB_UTR_t*,uint16_t,uint16_t)		
Arguments	USB_UTR_t*		Pointer to a USB Transfer Structure
	uint16_t		NOARGUMENT: Not used
	uint16_t		msginfo: Command category
Return values	―	―	―
Description	The following commands is set to the third argument USB_MSG_HCD_ATTACH: Executed at end of attach processing. USB_MSG_HCD_DETACH: Executed at end of detach processing. USB_MSG_HCD_SUSPEND: Executed at end of suspend processing. USB_MSG_HCD_RESUME: Executed at end of resume processing. USB_MSG_HCD_REMOTE: Executed at end of remote wakeup processing. USB_MSG_HCD_VBON: Executed at end of VBUS on processing. USB_MSG_HCD_VBOFF: Executed at end of VBUS off processing. USB_MSG_HCD_SETDEVICEINFO: Executed at end of pipe setting processing. USB_MSG_HCD_CTRL_END: Executed at end of data transfer. USB_MSG_HCD_CLRSEQBIT: Executed when sequence toggle bit is cleared. USB_MSG_HCD_SETSEQBIT: Executed when sequence toggle bit is set.		
Notes			

5.5.2 MGR call-backs

The follows shows MGR callback function.

Table5-5 classcheck

Name	Classcheck Call-back function		
Call format	(*USB_CB_CHECK_t)(USB_UTR_t*,uint16_t **)		
Arguments	USB_UTR_t*		Pointer to a USB Transfer Structure
	uint16_t	**table	table [0] device descriptor table [1] configuration descriptor table [2] interface descriptor table [3] descriptor check result table [4] hub type table [5] port number table [6] communication speed table [7] device address table [8] pipe information table
Return values	—	—	—
Description	HDCD is notified of descriptor information, etc., and HDCD returns a ready/not ready to operate result. One of the following check results (table[3]) are returned. USB_DONE: HDCD ready to operate USB_ERROR: HDCD not ready to operate. table[4] : HUB spec for HUB driver USB_FSHUB : Full-Speed HUB USB_HSHUBS : Hi-Speed HUB(Single) USB_HSHUBM : Hi-Speed HUB(Multi) table[5] : Port number (R8A66597 only) USB_PORT0 : Device connect to port0 USB_PORT1 : Device connect to port1 table[6] : Device speed (R8A66597 only) USB_HSCONNECT : Device is Hi-speed USB_FSCONNECT : Device is Full-speed USB_LSCONNECT : Device is Low-speed table[7] : DeviceAddress Device address which USB Host has assigned by SetAddress table[8] : Pipe information table		
Notes			

Table5-6 Other Callback function

Name	Other call-back functions		
Call format	(*USB_CB_INFO_t)(USB_UTR_t*,uint16_t ,uint16_t);		
Arguments	USB_UTR_t*	ptr	Pointer to a USB Transfer Structure
	uint16_t	devaddr	Connecte Device address
	uint16_t		NOARGUMENT: Not used
Return values	―	―	―
Description	The function set to the following memer is called at the registration. classinit: Executed at MGR startup. devconfig: Executed when Set_Configuration request issued. devdetach: Executed when a detach condition is detected. devsuspend: Executed at transition to suspend state. devresume: Executed at transition to resume state.		
Notes			

5.6 Structure Definitions

The structures used in USB host mode are described below. They are defined in file *usb_ctypedef.h*.

5.6.1 USB_HCDINFO_t structure

The USB_HCDINFO_t structure is used when transferring messages to HCD.
USB_HCDINFO_t is defined to "USB_UTR_t" type by *typedef*.

5.6.2 USB_HCDREG_t structure

USB_HCDREG_t is a structure for registering the information on HDCD. The call-back registered into this structure is called when the connected peripheral changes its device state.
Members of the USB_HCDREG_t structure are shown in Table5-7.

Table5-7 USB_HCDREG_t structure members

Type	Variable Name	Description
uint16_t	rootport	Used by HCD. The connected port number is registered.
uint16_t	devaddr	Used by HCD. The device address is registered.
uint16_t	devstate	Used by HCD. The device connection state is registered.
uint16_t	ifclass	Register the interface class code for HDCD operation.
uint16_t	*tpl	Register the target peripheral list for HDCD operation.
uint16_t	*pipetbl	Register the address of the pipe information table.
USB_CB_INFO_t	classinit	Function to be called when the driver is registered.
USB_CB_CHECK_t	classcheck	Register the HDCD "class check" function to be called when a TPL match occurs.
USB_CB_INFO_t	devconfig	Register the function to be started when transitioning to the configured state. It is called in the SET_CONFIGURATION request status stage.
USB_CB_INFO_t	devdetach	Register the function to be started when transitioning to the detach state.
USB_CB_INFO_t	devsuspend	Register the function to be started when transitioning to the suspend state.
USB_CB_INFO_t	devresume	Register the function to be started when transitioning to the resume state.

5.7 Target Peripheral List (TPL)

The USB-BASIC-F/W FIT module can specify USB device to support. User can specify the supported USB device by registering Vendor ID and Product ID of the USB device in the target peripheral list (TPL). If User does not have the supported USB device, please register USB_NOVENDOR and USB_NOPRODUCT.
Register the TPL as a member (tpl) in the USB_HCDREG_t structure. (See "5.6.2 USB_HCDREG_t structure")
A sample TPL is shown below.

```
const uint16_t usb_gapl_devicetpl[] =
{
    2,                  /* Number of list */
    0,                  /* Reserved */
    0xXXXX,             /* Vendor ID */
    0xXXXX,             /* Product ID */
    0xYYYY,             /* Vendor ID */
    0xYYYY,             /* Product ID */
};
```

[Note]

2 TPLs (for the application program and for USB Hub) are prepared in this F/W. When "USB Embedded Host compliance testing support setting" is selected, do not register USB_NOVENDOR and USB_NOPRODUCT in these 2 TPLs.
(Please refer to "5.12.1 **User Configuration File - r_usb_basic_config.h** (7)" about "USB Embedded Host compliance testing support setting".)

5.8 Pipe Information Table

In order for the HDCD to perform data communication on pipes 1 to 9, it must maintain information tables to define pipe settings. A pipe information table comprises the necessary register setting values for performing data transfers and a specification of the FIFO port transfer direction (CPU transfer or DTC transfer).

The follow shows the sample of the pipe information table.

```
uint16_t usb_gvendor_smpl_eptbl[] =
{
   USB_PIPE1,                                          ←Pipe definition item 1
   USB_NONE | USB_BFREOFF | USB_DBLBOFF |
   USB_CNTMDOFF | USB_SHTNAKOFF | USB_NONE,            ←Pipe definition item 2
   USB_BUF_SIZE(1024) | USB_BUF_NUMB(8),               ←Pipe definition item 3
   USB_NONE,                                           ←Pipe definition item 4
   USB_NONE,                                           ←Pipe definition item 5
   USB_CUSE,                                           ←Pipe definition item 6
         :
         :
   USB_PDTBEND,
}
```

A pipe information table comprises the following six items (uint16_t × 6).

1. Pipe Window Select register (address 0x64)
2. Pipe Configuration register (address 0x68)
3. Pipe Buffer Designation register (address 0x6A)
4. Pipe Maximum Packet Size register (address 0x6C)
5. Pipe Period Control register (address 0x6E)
6. FIFO port usage method

(1). Pipe definition item 1

Specify the value to be set in the pipe window select register.

Pipe select : Specify the selected pipe (USB_PIPE1 to USB_PIPE9)

Restrictions
—

(2). Pipe definition item 2

Specify the values to be set in the pipe configuration register.

Transfer type	:	Specify either USB_BULK or USB_INT
BRDY operation designation	:	Specify USB_BFREOFF
BRDY operation designation	:	Specify either USB_DBLBON or USB_DBLBOFF
Double buffer mode	:	Specify either USB_CNTMDON or USB_CNTMDOFF
SHTNAK operation designation	:	Specify either USB_SHTNAKON or USB_SHTNAKOFF
Transfer direction	:	Specify either USB_DIR_H_OUT or USB_DIR_H_IN
Endpoint number	:	Specify the endpoint number (EP1 to EP15)

Restrictions

- The values that can be set for the transfer type differ according to the selected pipe. For details, refer to the hardware manual of the corresponding device.
- For a pipe set to the receive direction, specify USB_SHTNAKON

(3). Pipe definition item 3

Only for USBA/USBAa. Specify the settings for the pipe buffer designation register.

Buffer size	:	Specify the pipe buffer size in 64-byte units
Buffer start number	:	Specify the buffer start number

Restrictions

- Make settings such that buffer areas in use at the same time do not overlap
- When the USB_DBLBON setting is used, twice the area is needed

(4). Pipe definition item 4

Specify the settings for the pipe maximum packet size register

Maximum packet size : Specify the maximum packet size for the pipe.

Restrictions

- The Max packet size which can be specified changes with devices. For details, please refer to the hardware manual of each device

Note

- When USB peripheral device with the maximum packet size MCU does not supports is connected, the data transfer is not operated normally.

(5). Pipe definition item 5

Specify the settings for the pipe period control register.

In-buffer flush	:	Specify USBC_IFISOFF
Interval duration	:	Specify the interval value (0 to 7).

Restrictions

- When a transmission type is set up in addition to USB_ISO, the ISO IN buffer flash should set up USB_IFISOFF.
- When selecting pipes 3 to 5, specify value 0 for the interval duration.

(6). Pipe definition item 6

Specify the FIFO port to be used for the pipe.

USB_CUSE	:	CFIFO is used and CPU access is carried out.
USB_D0USE	:	D0FIFO is used and CPU access is carried out.
USB_D0DMA	:	D0FIFO is used and DTC / DMA access is carried out.

USB_D1USE : D1FIFO is used and CPU access is carried out.
USB_D1DMA : D1FIFO is used and DTC / DMA access is carried out.

Restrictions
- A transaction counter starts the pipe of the receiving direction.
- No sample functions are provided for USB_D0USE and USB_D1DMA

(7). Other pipe setting notes
- Use device class to specify transfer unit communication synchronization.
- Do not fail to write USBC_PDTBLEND at the end of the table.

5.9 Host Control Transfer

The *R_usb_hstd_TransferStart*() function must be used to transmit not just standard requests ”5.2.2 USB Standard Requests” but also vendor or class requests.
To use R_usb_hstd_TransferStart() to transmit the setup packet, first set the following in the USB_UTR_t structure member, and then call R_usb_hstd_TransferStart().

- Set PIPE0 in*keyword
- Set USB_TRAN_END in *segment
- Set setup data in *setup

For more on the USB_UTR_t structure, refer to “5.10.4 The USB Communication Structure (USB_UTR_t）”
The following is an example of a vendor request transmit.

```
USBC_UTR_t    usb_gsmp_ControlSetupUtr;

USBC_ER_t     usb_hsmp_VendorRequestProcess(void)
{
   USBC_ER_t   err;

   /* Set Call-back function */
   usb_gsmp_ControlSetupUtr.complete     = &usb_smp_VendorRequestResult;

   /* Transmission pipe setup */
   usb_gsmp_ControlSetupUtr.keyword      = (uint16_t)USB_PIPE0;
   usb_gsmp_ControlSetupUtr.segment      = (uint8_t)USBC_TRAN_END;
   /* Setup packet data */
   usb_gsmp_ControlSetupUtr.tranadr      = (void*)usb_gsmp_VendorRequestData;
   /* Transfer size setup */
   usb_gsmp_ControlSetupUtr.tranlen      = (uint32_t)usb_gsmp_tranlen;
   /* Setup command setup */
   usb_gsmp_ControlSetupUtr.setup        = (uint16_t*)&usb_gsmp_VendorRequest_Table;

   /* Data Request to Send */
   err = R_usb_hstd_TransferStart(&usb_gsmp_ControlSetupUtr);
   return err;
}
```

5.10 Data Transfer and Control Data Transfer

5.10.1 Data transfer request

A data transfer demand is made by using R_usb_hstd_TransferStart().. For more details on the API functions described here, please refer to the corresponding API section.

5.10.2 Notification of transfer result

When a data transfer completes, the firmware notifies the HDCD of the data transfer completion and the sequence toggle bit information in the call-back function registered when the data transfer was requested.
The results of the transfer are stored in the USB_UTR_t structure member (status).
Below, a communication result is shown.

USB_CTRL_END	:	Control transfer normal end
USB_DATA_NONE	:	Data transmit normal end
USB_DATA_OK	:	Data receive normal end
USB_DATA_SHT	:	Data receive normal end with less than specified data length
USB_DATA_OVR	:	Receive data size exceeded
USB_DATA_ERR	:	No-response condition or over/under run error detected
USB_DATA_STALL	:	STALL response or MaxPacketSize error detected
USB_DATA_STOP	:	Data transfer forced end
USB_DATA_TMO	:	Forced end due to timeout, no call-back (RTOS only)

5.10.3 Data transfer retry

When a no-response condition occurs on a pipe, USB-BASIC-F/W performs communication retries up to the user-specified number of times. If the no-response condition continues through the end of the user-specified number of retries, the notification USB_DATA_ERR is returned to HDCD.

5.10.4 The USB Communication Structure (USB_UTR_t)

The structure used for Control and data transfer is described below. USB communications with the connected device are enabled by notifying HCD of the following structure members.
Members of USB_UTR_t Structure are shown in Table5-8.

Table5-8 USB Communication Structure members and description for use with host

Type	Member	Description
USB_MH_t	msghead	This message header is used by the OS (or non-OS messaging system). It should not be used by the application
uint16_t	msginfo	Specifies the request content. Specifies USB-BASIC-F/W using an API function. Specifies USB_MSG_HCD_SUBMITUTR for USB communication.
uint16_t	keyword	Please register sub information, including a port number, a pipe number, etc. A pipe number is specified when performing USB communication.
USB_REGADR_t	ipp	Register the address of USB IP.
uint16_t	ip	Register the number of USB IP.
uint16_t	result	Store the USB communication result.
USB_CB_t	complete	Callback function Specifies the address of the function to be executed when USB transfer ends. Use the following type declaration for the call-back function. typedef void (*USB_CB_t)(USB_UTR_t *);
void	*tranadr	Buffer address for transfer data; where to read data to send, or where to write receive-data.
uint32_t	tranlen	Specifies nr of bytes to transfer when transmitting, and provides information on nr bytes received when receiving. Specify a transfer size less or equal the user buffer size.
uint16_t	*setup	Setup packet data. Requests a setup packet for control transfer to HCD and provides notification of the device address. Only used for control transfer
uint16_t	status	USB communication status. HCD returns the USB communication result. Refer to "「5.10.2 Notification of transfer result."
uint16_t	pipectr	PIPEnCTR register. HCD returns the PIPEnCTR register information for pipe n. When using one pipe for two endpoints, specify the transfer toggle state for the endpoint when continuing the communication.
uint8_t	errcnt	Stores the number of errors that occur during transfer.
uint8_t	segment	Segment information. Provides notification of transfer continuation/end.
void	*usr_data	Sets each type of data.

(1). Buffer address for USB communication (tranadr)

Reception or ControlRead transfer: Specifies the address of the buffer for storing receive data.

Transmission or ControlWrite transfer: Specifies the address of the buffer for storing transmit data.

NoDataControl transfer: Ignored if specified.

(2). USB communication data length (tranlen)

Reception or ControlRead transfer: Stores the receive data length.

Transmission or ControlWrite transfer: Stores the transmit data length.

NoDataControl transfer: Set to 0.

The remaining transmit/receive data length is stored after USB communication end. In case of control transfer in host mode, the remaining data length from the data stage is stored.

(3). Setup packet data (setup)

For control transfers using *R_usb_hstd_TransferStart*(), the structure member (*setup) is a USB request data table stored in the hardware registers listed below.

For *setup, specify the table address of the uint16_t[5] array.

Table5-9 setup_packet array

Register	Name	Value	
54H	USBREQ	bRequest	bmRequestType
56H	USBVAL	wValue	
58H	USBINDX	wIndex	
5AH	USBLENG	wLength	
--	USBADDR	Device Address	

(4). Pipe control (pipectr)

An HDCD driver can communicate with multiple endpoint addresses over a single pipe by remembering the sequence toggle bit. When specifying transfer continuation, set the SQMON bit in this register to the previous toggle state.

(5). Segment information (pipectr)

Control transfer continuation: Specify USBC_TRAN_CONT (continuation of transfer from data stage enabled).
Control transfer end: Specify USBC_TRAN_END.
Data transfer continuation: Specify USBC_TRAN_CONT. (Set SQMON bit in PIPECTR.)
Data transfer end: Specify USBC_TRAN_END. During isochronous transfer no determination is made when pipe is in use.

5.10.5 Notes on Data Transfer

When the maximum packet size is an odd number and one packet is not equivalent to one transfer, the CPU may generate an address exception during buffer access.

5.10.6 Notes on Data Reception

Use a transaction counter for the receive pipe.
When a short packet is received, the expected remaining receive data length is stored in tranlen of USB_UTR_t structure and transfer ends. When the received data exceeds the buffer size, data read from the FIFO buffer up to the buffer size and transfer ends. When the user buffer area is insufficient to accommodate the transfer size, the usb_cstd_DataEnd() function may clear the receive packet in some cases.
When using DTC, the buffer size must be an integral multiple of the maximum packet size.

5.10.7 Data Transfer Overview

The following shows an overview of data transfer using a transmission as an example.
(1). Transfer status is set by the APL or HDCD in the USB_UTR_t structure members listed below.
- keyword: Pipe number
- tranadr: Data buffer
- tranlen: Transfer length
- segment: USB_TRAN_END
- complete: The callback function performed at the time of the end of data transfer

(2). R_usb_hstd_TransferStart () is called and data transfer request executed.
(3). After data transfer is complete, the call-back function “complete” that was set in (1) is called and the transfer results are notified to HDCD. (See 5.10.2 Notification of transfer result.)

5.11 Host Battery Charging (HBC)

HBC is the H/W control program for the target device that operates the CDP or the DCP as defined by the USB Battery Charging Specification Revision 1.2.
Processing is executed as follows according to the timing of the USB-BASIC-F/W. Refer to Figure 5-1.

VBUS is driven
Attach processing
Detach processing

Moreover, processing is executed in coordination with the PDDETINT interrupt.
There is no necessity for control from the upper layer.
The USB-BASIC-F/W notifies the result of the CPD action to the upper layer by the callback function which is defined to USB_BC_ATTACH macro, using the second argument.
The result of the callback notified to the upper layer is one of the following:

2 : Charging Downstream Port (CDP) Detection
3 : Standard Downstream Port (SDP) Detection
4 : Dedicated Charging Port (DCP) Detection

CDP and DCP exclude other execution of the Basic FW. When DCP is operating, USB communication cannot be done.

The processing flow of HBC is shown Figure 5-1.

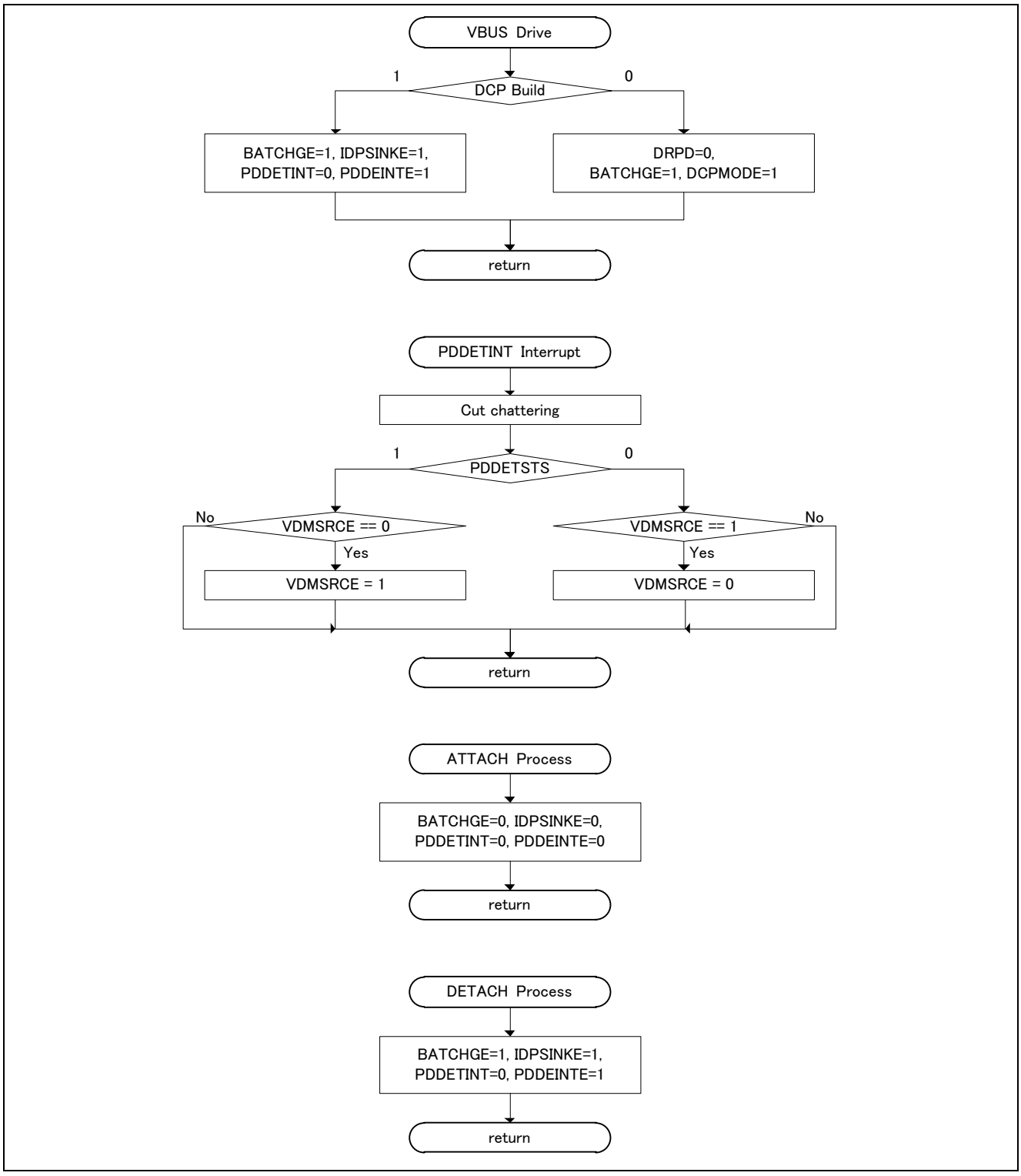

Figure 5-1 HBC processing flow

5.12 How to operate USB-BASIC-F/W as Host mode

This section describes the procedure for operating the USB-BASIC-F/W FIT module in Host mode.

5.12.1 User Configuration File - r_usb_basic_config.h

Functional settings for the USB-BASIC-F/W module are specified by overwriting the user-defined information file (r_usb_basic_config.h). Modify the items below to match the specifications of the system.

(1). DTC FIT module setting

Selects whether or not the DTC FIT module is used.

```
#define USB_DTC_ENABLE      : DTC FIT module used
#define USB_DTC_ENABLE      : DTC FIT module not used
```

(2). USB operating mode (Host / Peripheral) setting

Specifies the operating mode (Host mode) per the USB module (USBb, USBA/USBAa). If USB module is not used, specifies USB_NOUSE_PP.

```
#define USB_FUNCSEL_USBIP0_PP   USB_HOST_PP     : Host mode for USBb
#define USB_FUNCSEL_USBIP0_PP   USB_NOUSE_PP    : USBb not used
#define USB_FUNCSEL_USBIP1_PP   USB_HOST_PP     : Host mode for USBA/USBAa
#define USB_FUNCSEL_USBIP1_PP   USB_NOUSE_PP    : USBA/USBAa not used
```

(3). Specifies the USB module type of USB module (USBA/USBAa).

```
#define USB_SPEED_PP    USB_HS_PP :Hi-speed module type setting of USBA/USBAa module
#define USB_SPEED_PP    USB_FS_PP :Full-speed module setting of USBA/USBAa module
```

[Note]

Be sure to set USB_FS_PP when using RX64M.

(4). Battery Charging functionality in Host mode

Specifies whether or not battery charging functionality is enabled during host operation. To use Battery Charging functionality, enable this macro. This item is valid when the setting of USB_FUNCSEL_USBIP0_PP / USB_FUNCSEL_USBIP1_PP is USB_HOST_PP.

```
#define USB_HOST_BC_ENABLE      : Battery Charging function used
#define USB_HOST_BC_ENABLE      : Battery Charging function not used
```

(5). USB port operation when using battery charging functionality setting

Specifies whether or not the USB module functions as a dedicated charging port (DCP) when battery charging functionality is enabled during host operation. To use the USB module as a DCP, enable this macro.This item is valid when the setting of USB_FUNCSEL_USBIP0_PP / USB_FUNCSEL_USBIP1_PP is USB_HOST_PP.

```
#define USB_BC_DCP_ENABLE       : Dedicated Charging Port（DCP）selected
#define USB_BC_DCP_ENABLE       : Charging Downstream Port（CDP）selected
```

(6). Device attach function setting

Register the function that is called when a device is attached during host operation. This item is valid when Battery Charging functionality is used

```
#define USB_BC_ATTACH(ptr, data1, data2)        usb_cstd_DummyFunction(ptr, data1, data2)
```

(7). USB Embedded Host compliance testing support setting

Specifies whether or not USB embedded host compliance testing is supported during host operation. To enable support for compliance testing, enable this macro.

```
#define USB_HOST_COMPLIANCE_MODE    : Compliance test mode enable
//#define USB_HOST_COMPLIANCE_MODE  : Compliance test mode disable
```

(8). Compliance test information display function setting

Register the function for displaying information on the display device (LCD, etc.) when compliance testing is enabled.

#define USB_COMPLIANCE_DISP(ptr, data1, data2) usb_cstd_DummyFunction(ptr, data1, data2)

(9). Compliance test response time setting

Specify in ms units the time that must elapse before the device is determined to have returned no response to a control transfer.This item is valid when compliance testing is enabled.

#define USB_RESPONCE_COUNTER_VALUE (6000u)

(10). Hi-speed electrical test setting

Enables this definition when doing Hi-speed electrical test in enabling USB_HOST_COMPLIANCE_MODE.

#define USB_HS_EL_TEST : Hi-speed electrical test enable

// #define USB_HS_EL_TEST : Hi-speed electrical test disable

(11). FIFO access endian setting

Specify the endian mode for FIFO access.

#define USB_CPUBYTE_PP USB_BYTE_LITTLE_PP : Little endian
#define USB_CPUBYTE_PP USB_BYTE_BIG_PP : Big endian

(12). Low-power mode setting

Specify whether or not low-power mode is used.

#define USB_CPU_LPW_PP USB_LPWR_USE_PP : Low Power Mode used
#define USB_CPU_LPW_PP USB_LPWR_NOT_USE_PP : Low Power Mode not used

(13). Overcurrent detection function setting

Register the function that is called when overcurrent detection occurs in Host mode.

USB_OVERCURRENT(ptr, data1, data2) usb_cstd_DummyFunction(ptr, data1, data2)

(14). Task etc ID maximum value and Task priority value

Set the value to the following definition depending on User system. RAM size changes by this value.

a) USB_IDMAX
b) USB_PRIMAX
c) USB_BLKMAX

6. HUB Class

6.1 Basic Functions

HUBCD is a task that manages the states of the down ports of a connected USB hub and supplements the functions of HCD and HDCD. HUBCD performs the following functions.

1. State management for devices connected to the USB hub down ports
2. Enumeration of devices connected to the USB hub down ports

6.2 HUBCD API Functions

Table6-1 lists the API functions of HUBCD.

Table6-1 List of HUBCD API Functions

Function Name	Description
R_usb_hhub_Registration	Register HUBCD
R_usb_hhub_Task	HUB Task

6.2.1 R_usb_hhub_Registration

Register HUBCD

Format

void	R_ usb_hhub_Registration(USB_UTR_t *ptr, USB_HCDREG_t *p_hcdreg)

Argument

*ptr	Pointer to a USB Transfer Structure
*callback	Pointer to USB_HCDREG_t structure

Return Value

—	—

Description

The information on HUBCD is registered.

Note

1. Besides above arguments, also set the following members of the USB Transfer Structure.
   USB_REGADR_t ipp :USB register base address
   uint16_t ip :USB IP number
2. The user must call this function from the user application during initialization.
3. Call this API after he start address of TPL area is set to the member ‘tpl’ and the start address of the pipe information table is set to the member ‘pipetbl’. Refer to Table5-7 USB_HCDREG_t structure members for details about registration information.

Example

```
extern uint16_t  usb_ghhub_TPL[];
extern uint16_t  usb_ghhub_DefEPTbl[];;
void  usb_smp_registration(USB_UTR_t *ptr)
{
  USB_HCDREG_t  driver;
    :
  driver.tpl = usb_ghhub_TPL;
  driver.pipetbl = usb_ghhub_DefEPTbl;
  R_usb_hhub_Registration(ptr, &driver);
    :
}
```

6.2.2 R_usb_hhub_Task

HUB Task

Format

void R_usb_hhub_Task(USB_VP_INT stacd)

Argument

stacd Task start code (Not used)

Return Value

— —

Description

At the time of selection of a host, the state of the down port of the connected USB hub is managed, and the function between HCD and HDCD is complemented.

Note

1. Call this in the loop that executes the scheduler processing

Example

```
void  usb_smp_mainloop(void)
{
  while(1)
  {
  /* Scheduler starting */
  R_usb_cstd_Scheduler();
  /* Flag check */
  if(USB_FLGSET == R_usb_cstd_CheckSchedule())
  {
       :
   R_usb_hhub_Task ((USB_VP_INT)0);
       :
  }
    :
}
```

6.3 Down Port State Management

When the HUBCD task starts, it performs the following for all down ports and confirms that each down port is in the device connected state.

(1). Enable port power (HubPortSetFeature:USB_HUB_PORT_POWER)
(2). Initialize port (HubPortClrFeature:USB_HUB_C_PORT_CONNECTION)
(3). Get port status (HubPortStatus:USB_HUB_PORT_CONNECTION)

6.4 Connecting Devices to Down Ports

When HUBCD receives a down port device attach notification from the USB hub, it issues a USB reset signal request to the USB hub (HubPortSetFeature: USB_HUB_PORT_RESET). Then it uses the MGR task resource to enumerate the connected devices. HUBCD stores the reset handshake result and assigns addresses successively, starting from USBC_DEVICEADDR + 1.

6.5 Class Requests

The class request which HUBCD is supporting is shown in Table6-2.

Table6-2 USB Hub Class Requests

Request	Installation Status	Function Name	Description
ClearHubFeature	×		
ClearPortFeature	○	usb_hhub_PortClrFeature	USB_HUB_PORT_ENABLE USB_HUB_PORT_SUSPEND USB_HUB_C_PORT_CONNECTION USB_HUB_C_PORT_ENABLE USB_HUB_C_PORT_SUSPEND USB_HUB_C_PORT_OVER_CURRENT USB_HUB_C_PORT_RESET
ClearTTBuffer	×		
GetHubDescriptor	○	R_usb_hhub_GetHubInformation	Get descriptor
GetHubStatus	×		
GetPortStatus	○	R_usb_hhub_GetPortInformation	Get port status
ResetTT	×		
SetHubDescriptor	×		
SetHubFeature	×		
SetPortFeature	○	usb_hhub_PortSetFeature	USB_HUB_PORT_POWER USB_HUB_PORT_RESET USB_HUB_PORT_SUSPEND USB_HUB_C_PORT_ENABLE
GetTTState	×		
StopTT	×		

7. non-OS Scheduler

7.1 Overview

The non-OS scheduler executes task scheduling according to task priority. The following are non-OS task scheduler features.

- A scheduler function manages requests generated by the tasks and interrupt etc according to the relative priority of the tasks.
- When multiple requests are generated by tasks with the same priority, they are executed using a FIFO configuration.
- Call-back functions are used for responses to tasks indicating the end of a request, so the customer need only install appropriate class drivers for the system and there is no need to modify the scheduler itself.

7.2 non-OS Scheduler Macro

Table7-1 shows the scheduler macro that user uses.

Table7-1 Scheduler macro

Scheduler macro	Registration function	Description
R_USB_SND_MSG	R_usb_cstd_SndMsg	Specifies Task ID and transmits message
R_USB_ISND_MSG	R_usb_cstd_iSndMsg	Specifies Task ID from interrupt and transmits message.
R_USB_WAI_MSG	R_usb_cstd_WaiMsg	Runs USB_SND_MSG after running the scheduler the specified number of times.
R_USB_RCV_MSG	R_usb_cstd_RecMsg	Checks if received message is in specified mailbox.
R_USB_TRCV_MSG	R_usb_cstd_RecMsg	Checks if received message is in specified mailbox.
R_USB_PGET_BLK	R_usb_cstd_PgetBlk	Secures an area for storing a message.
R_USB_REL_BLK	R_usb_cstd_RelBlk	Releases an area for storing a message.

7.2.1 non-OS Scheduler API Function

Table 7-2 shows the list of API function for non-OS scheduler.

Table 7-2 List of non-OS scheduler API function

API function	Description
R_usb_cstd_SndMsg()	Sends processing requests to the priority table.
R_usb_cstd_iSndMsg()	Sends processing requests from interrupts to the priority table.
R_usb_cstd_RecMsg()	Checks if an execution request was issued.
R_usb_cstd_PgetBlk()	Allocates an area for storing a request information table.
R_usb_cstd_RelBlk()	Releases an area used for storing a request information table.
R_usb_cstd_Scheduler()	Manages requests generated by the tasks.
R_usb_cstd_SetTaskPri()	Set Task Priority.
R_usb_cstd_CheckSchedule()	Checks whether or not processing is scheduled.

7.2.2 R_usb_cstd_SndMsg

Sends processing requests to the priority table

Format

USB_ER_t	R_usb_cstd_SndMsg(uint8_t id, USB_MSG_t *mess)

Argument

id	Task ID to send message
*mess	Transmitted message

Return Value

USB_E_OK	Message transmission completion
USB_E_ERROR	Task ID is not set Priority of task is not set Priority table is full (Can't send request to priority table)

Description

This function transfers the processing request to the priority table.

The function is defined in the R_USB_SND_MSG macro.

Note

1. Please call the sceduler macro that this function is defined in theuser application other than the interrupt function or the class driver.
2. Specify the start address of the memory buffer in the message to be transmitted.
3. The sample program specifies the start address of the memory block obtained in the R_PGET_BLK macro.

Example

```
void  usb_smp_task()
{
  USB_MH_t         p_blf;
  USB_ER_t         err;
  USB_PCDINFO_t *pp;
    :
  /* Allocating the messsage store area */
  err = R_USB_PGET_BLK(USB_PVENDOR_MPL, &p_blf);
  if(err == USB_OK)
  {
    /* Setting Message */
    pp = (USB_CLSINFO_t*)p_blf;
    pp->msghead    = (USB_MH_t)NULL;
    pp->msginfo    = USB_SMP_REQ;
    pp->keyword    = keyword;
    pp->complete   = complete;
    pp->ip         = ptr->ip;
    pp->ipp        = ptr->ipp;
  }
  /* Send the message */
  err = R_USB_SND_MSG(USB_SMP_MBX, (USB_MSG_t*)cp);
  if(err != USB_OK)
  {
    /* Error processing */
  }
    :
}
```

7.2.3 R_usb_cstd_iSndMsg

Sends processing requests from interrupts to the priority table

Format

USB_ER_t	R_usb_cstd_iSndMsg(uint8_t id, USB_MSG_t *mess)

Argument

id	Task ID to send message
*mess	Transmitted message

Return Value

USB_E_OK	Message transmission completion
USB_E_ERROR	Task ID is not set Priority of task is not set Priority table is full (Can't send request to priority table

Description

This function transfers the processing request to the priority table.

The function is defined in the R_USB_ISND_MSG macro.

Note

1. Please call the sceduler macro that this function is defined in the interrupt function.
2. Specify the start address of the memory buffer in the message to be transmitted.
3. The sample program specifies the start address of the memory block obtained in the R_PGET_BLK macro.

Example

```
void  usb_smp_interrupt()
{
  USB_MH_t         p_blf;
  USB_ER_t         err;
  USB_PCDINFO_t *pp;
    :
  /* Allocating the messsage store area */
  err = R_USB_PGET_BLK(USB_PVENDOR_MPL, &p_blf);
  if(err == USB_OK)
  {
    /* Setting Message */
    pp = (USB_CLSINFO_t*)p_blf;
    pp->msghead    = (USB_MH_t)NULL;
    pp->msginfo    = USB_SMP_REQ;
    pp->keyword    = keyword;
    pp->complete   = complete;
    pp->ip         = ptr->ip;
    pp->ipp        = ptr->ipp;
  }
  /* Send the message */
  err = R_USB_ISND_MSG(USB_SMP_MBX, (USB_MSG_t*)pp);
  if(err != USB_OK)
  {
    /* Error processing */
  }
    :
}
```

7.2.4 R_usb_cstd_RecMsg

Checks if an execution request was issued

Format

USB_ER_t R_usb_cstd_RecMsg(uint8_t id, USB_MSG_t** mess, USB_TM_t tm)

Argument

id	Task ID of received message
*mess	Received message
tm	Not Used

Return Value

USB_E_OK	There is request processing
USB_E_ERROR	Task ID is not set

Description

This function checks if the processing request corresponding to the ID specified in the 1st argument is in the priority table.

The function is defined in the R_USB_RCV_MSG/ R_USB_TRCV_MSG macro.

Note

1. Please call the sceduler macro that this function is defined from the user application or the class driver.
2. When the return value of R_usb_cstd_CheckSchedule is USB_FLGCLR, do not call this API.

Example

```
void  usb_smp_task()
{
  USB_UTR_t        *mess;
  USB_ER_t         err;
    :
  /* Checking the message */
  err = R_USB_RCV_MSG(USB_SMP_MBX, (USB_MSG_t**)&mess);
  if(err == USB_OK)
  {
    /* Processing corresponding to the received message */
  }
    :
}
```

7.2.5 R_usb_cstd_PgetBlk

Allocates an area for storing a request information table

Format

USB_ER_t	R_usb_cstd_PgetBlk(uint8_t id, USB_UTR_t **blk)

Argument

id	Task ID of allocating area
**blk	Pointer to allocating area

Return Value

USB_E_OK	The area is allocated
USB_E_ERROR	Task ID is not set
	The area is not secured

Description

This function secures an area for storing a message. The function allocates 40 bytes per block.

The function is defined in the R_USB_PGET_BLK macro.

Note

1. Please call the sceduler macro that this function is defined from the user application or the class drive.

Example

```
void  usb_smp_task()
{
  USB_ER_t         err;
  USB_PCDINFO_t *pp;
    :
  /* Allocating the are that the message is stored */
  err = R_USB_PGET_BLK(USB_SMP_MPL, &p_blf);
  if(err == USB_OK)
  {
    /* Setting Message */
    pp = (USB_CLSINFO_t*)p_blf;
    pp->msghead    = (USB_MH_t)NULL;
    pp->msginfo    = USB_SMP_REQ;
    pp->keyword    = keyword;
    pp->complete   = complete;
    pp->ip         = ptr->ip;
    pp->ipp        = ptr->ipp;
    /* Send the message */
    err = R_USB_ISND_MSG(USB_SMP_MBX, (USB_MSG_t*)pp);
  }
  else
  {
    /* Error Processing */
  }
    :
}
```

7.2.6 R_usb_cstd_RelBlk

Releases an area used for storing a request information table

Formant

USB_ER_t	R_usb_cstd_RelBlk(uint8_t id, USB_UTR_t* blk)

Argument

id	Task ID of releasing area
*blk	Pointer to released area

Return Value

USB_E_OK	The area is released
USB_E_ERROR	Task is not set
	The area is not released

Description

This function releases an area for storing a message.

The function is defined in the R_USB_REL_BLK macro.

Note

1. Please call the sceduler macro that this function is defined from the user application or the class drive.

Example

```
void  usb_smp_task()
{
  USB_ER_t         err;
  USB_UTR_t        *mess;
    :
  /* Releasing the area that the message is stored */
  err = R_USB_REL_BLK(USB_SMP_MPL,(USB_MH_t)mess);
  if(err != USB_OK)
  {
    /* Error processing */
  }
    :
}
```

7.2.7 R_usb_cstd_Scheduler

Manages requests generated by the tasks

Format

void	R_usb_cstd_Scheduler(void)

Argument

—	—

Return Value

—	—

Description

Managing requests generated by the tasks and hardware according to the relative priority of the tasks

Note

1. Call this function in the loop that executes the scheduler processing for non-OS operations.

Example

```
void  usb_smp_mainloop(void)
{
  while(1)
  {
  /* Check flag */
  if(USB_FLGSET == R_usb_cstd_CheckSchedule())
  {
       :
    /* Task Processing */
       :
  }
  /* Working Scheduler */
  R_usb_cstd_Scheduler();
       :
}
```

7.2.8 R_usb_cstd_SetTaskPri

Set Task Priority

Format

void	R_usb_cstd_SetTaskPri(uint8_t tasknum, uint8_t pri)

Argument

tasknum	Task ID
pri	Task Priority

Return Value

—	—

Description

Set the priority to the task. This function sets the priority of the task. Priority (pri) specified in 2nd argument is set to the task specified in the 1st argument (tasknum).

Note

1. Please call the sceduler macro that this function is defined from the user application or the class drive.
2. Please refer to ‘8 How to register in USB-BASIC-F/W’ about Task priority.setting

Example

```
void usb_smp_driver_start(void)
{
  /* The priority of the sample task is set to 4 */
  R_usb_cstd_SetTaskPri(USB_SMP_TSK, USB_PRI_4);
}
```

7.2.9 R_usb_cstd_CheckSchedule

Checks whether or not processing is scheduled

Format

uint8_t R_usb_cstd_CheckSchedule(void)

Argument

— —

Return Value

USB_FLGSET	Processing request
USB_FLGCLR	No processing request

Description

Checks whether or not processing is scheduled.

Note

1. Call this in the loop that executes the scheduler processing

Example

```
void  usb_smp_mainloop(void)
{
  while(1)
  {
  /* Flag Check */
  if(USB_FLGSET == R_usb_cstd_CheckSchedule())
  {
       :
   /* Task Processing */
       :
  }
  /* Working Scheduler */
  R_usb_cstd_Scheduler();
       :
}
```

8. How to register in USB-BASIC-F/W

The following settings are required to register class drivers and applications in non-OS.
For details on schedulers used by non-OS, refer to chapter 7. non-OS Scheduler.

1. Scheduler settings
2. Task information settings
3. Task registrations in scheduler
4. Task calls

(1). Setup of scheduler

Set the following items in USB-BASIC-F/W r_usb_cKernelid.h file.

#define	USB_IDMAX	Maximum number*1 of task IDs
#define	USB_TABLEMAX	Maximum number of messages storable in the priority table for each task priority
#define	USB_BLKMAX	Maximum number of information tables used to send messages from each task

*1: For the maximum number setting, add 1 to the highest ID number among the tasks to be used.

(2). Setup of task information

For each additional task, add the task ID, mailbox ID, memory pool ID and task priority to the USB-BASIC-F/W r_usb_ckernelid.h file.
Keep the following points in mind when setting these items.

- Assign consecutive ID numbers to the tasks, and do not assign the same ID to more than one task.
- Set the same value assigned to task ID for mailbox ID and memory pool ID.
- Set the task priority of added class drivers and applications to a value lower than that of PCD.

The following settings are examples for vendor class drivers of the sample program.

#define	USB_PVENDOR_TSK	USB_TID_6	: Task ID
#define	USB_PVENDOR_MBX	USB_PVENDOR_TSK	: Mailbox ID
#define	USB_PVENDOR_MPL	USB_PVENDOR_TSK	: Memory pool ID

(3). Task registration to a scheduler

R_usb_cstd_SetTaskPri() is used to set task ID priorities in the scheduler.
Set the task ID in the 1st argument of R_usb_cstd_SetTaskPri() and the task priority in the 2nd argument.

(4). Calling the application task

The following shows the point of calling the application task.

a). If the application task uses non-OS scheduler API

Please call the application task at the "Inside of the scheduler management" or "outside of the sckeduler management".

Example)

```
while(1)
{
    /* Scheduler processing */
    R_usb_cstd_Scheduler();
    if(USB_FLGSET == R_usb_cstd_CheckSchedule())
    {
        /* Inside of the scheduler management */
        R_usb_pstd_PcdTask((USB_VP_INT)0);      /* PCD Task */
        R_usb_pvendor_task((USB_VP_INT)0);      /* PDCD Task */
        usb_pvendor_apl_task((USB_VP_INT)0);    /* APL Task */
    }
    /* Outside of the scheduler management */
}
```

b). If the application task does not use non-OS scheduler API
Please call the application task at the "outside of the sckeduler management".
Example)

```
while(1)
{
    /* Scheduler processing */
    R_usb_cstd_Scheduler();
    if(USB_FLGSET == R_usb_cstd_CheckSchedule())
    {
       R_usb_pstd_PcdTask((USB_VP_INT)0);   /* PCD Task */
       R_usb_pvendor_task((USB_VP_INT)0);   /* PDCD Task */
    }
    /* Outside of the scheduler management */
    usb_pvendor_apl_task((USB_VP_INT)0);    /* APL Task */
}
```

9. DTC Transfer

9.1 Overview

When D0FIFO access (USBC_D0DMA) is specified in the pipe information table, USB-BASIC-F/W uses the DTC for FIFO access. Note that the DTC is a system-dependent function, so the settings for transfer method, access timing, communication start/end timing, DTC, etc., should be changed as necessary to match the system under development.

9.1.1 Basic Specification

The specifications of the DTC transfer sample program code included in USB-BASIC-F/W are listed below. DTC settings are made by the usb_cpu_d0fifo2buf_start_dma (),usb_cpu_buf2d0fifo_start_dma () function in the rx_rsk.c file. When DTC is specified in the pipe information table, the usb_cstd_SendStart() or usb_cstd_ReceiveStart() function makes DTC transfer settings.
Modify the DTC control sample functions usb_cpu_d0fifo2buf_start_dma () and usb_cpu_buf2d0fifo_start_dma () and the DTC settings as necessary to match the system under development.
And, pipe 1 to pipe5 can used DTC access.
Table9-1 shows DTC Setting Specifications.

Table9-1 DTC Setting Specifications

Setting	Description
FIFO port used	D0FIFO port
Transfer mode	Block transfer mode One DTC transfer size: max packet size.
Chain transfer	Disabled
Address mode	Full address mode
Read skip	Disabled
Access bit width (MBW)	2-byte transfer: 16-bit width 1-byte transfer: 8-bit width (In data reception (D0FIFO to RAM), this setting is 16-bit width.)
Transfer end	Receive direction: BRDY interrupt Transmit direction: BEMP interrupt

9.1.2 Notification of Transfer Result

When DTC access is used as well, USB-BASIC-F/W uses the registered call-back function(this function is specified by R_usb_hstd_TransferStart/R_usb_pstd_TransferStart) to notify PDCD/HDCD of the data transfer end condition.
USB-BASIC-F/W can send the following seven communication result notifications to PDCD/HDCD.
Transfer Result is set to the member “status ” of USB_UTR_t structure.
The following shows Transfer Result

USB_DATA_NONE	:	Data transmit normal end
USB_DATA_OK	:	Data receive normal end
USB_DATA_SHT	:	Data receive normal end with less than specified data length
USB_DATA_OVR	:	Receive data size exceeded
USB_DATA_ERR	:	No-response condition or over/underrun error detected
USB_DATA_STALL	:	STALL response or MaxPacketSize error detected
USB_DATA_STOP	:	Data transfer forced end

9.1.3 Notes on Data Reception

When the received data exceeds the buffer size, data is read from the FIFO buffer up to the buffer size and transfer ends.
In Host mode, when the pipe information table is updated in the HCD API function, the following settings are implemented.
Please carry out the following setup to a pipe information table at the time of Peripheral operation.

- Enable the SHTNAK function for the data receive pipe.
- The data receive pipe processes requests with the BFRE function enabled. (When the Ep table is set to USB_D0DMA = DTC transfer, the pipe configuration is set to BFRE ON.)
- Use the transaction counter for the data receive pipe.

The area of integral multiples of the max packet size is necessary for the buffer where the received data is stored. When the max packet size is 64 bytes, the necessary size of buffer example is shown to the receive data size below.

Table9-2 Buffer size example (max packet size is 64bytes)

Receive data size	Buffer size [bytes]
64 bytes or less	64 (max packet size)
65 bytes or more and 128 bytes or less	128 (max packet size times two)
…	…
449 bytes or more and 512 bytes or less	512 (max packet size times eight)
…	…

9.2 How to DTC transfer in the sample program

The Ep table must be edited in order to execute a DTC transfer in the sample application.

9.2.1 Pipe information table setting for DTC transfer in the Host sample program

Edit Ep table “usb_shvendor_smpl_tmp_eptbl” which is in the “r_usb_hvendor_defep.c”.

```
uint16_t usb_shvendor_smpl_tmp_eptbl[] =
{
        /* PIPE1 */
        USB_NONE,
        /* TYPE/BFRE/DBLB/CNTMD/SHTNAK/DIR/EPNUM */
        USB_NONE | USB_BFREOFF | USB_DBLBON | USB_CNTMDON | USB_SHTNAKON | USB_NONE | USB_NONE,
        USB_NONE,
        USB_NONE,
        USB_NONE,
        USB_CUSE,

        /* PIPE2 */
        USB_NONE,
        /* TYPE/BFRE/DBLB/CNTMD/SH
        USB_NONE | USB_BFREOFF | USB_DBLBON | USB_CNTMDON | USB_SHTNAKON | USB_NONE | USB_NONE,
        USB_NONE,
        USB_NONE,
        USB_NONE,
        USB_CUSE,
```

When DTC is used in pipe 1 (BULK IN: receive), USB_CUSE is changed to USB_D0DMA.

When DTC is used in pipe 2 (BULK OUT: transfer), USB_CUSE is changed to USB_D0DMA.

[Note]

1. Either pipe 1 or pipe 2 can be used to run DTC. However, both pipes cannot be used to run DTC.

9.2.2 Pipe information table setting for DTC transfer in the Peripheral sample program

Edit Ep table “usb_gpvendor_smpl_eptbl1” which is in the “r_usb_vendor_descriptor.c”.

```
uint16_t usb_gpvendor_smpl_eptbl1[] =
{
        /* PIPE1 Definition */
        USB_PIPE1,
        USB_BULK | USB_BFREOFF | USB_DBLBON | USB_SHTNAKON | USB_DIR_P_IN    | USB_EP1,
        USB_NONE,
        64,
        USB_IFISOFF | USB_IITV_TIME(0u),
        USB_CUSE,

        /* PIPE2 Definition */
        USB_PIPE2,
        USB_BULK | USB_BFREOFF | USB_
        USB_NONE,
        64,
        USB_IFISOFF | USB_IITV_TIME(0u),
        USB_CUSE,
```

When DTC is used in pipe 1 (BULK IN: receive), USB_CUSE is changed to USB_D0DMA.

When DTC is used in pipe 2 (BULK OUT: transfer), USB_CUSE is changed to USB_D0DMA.

[Note]

1. Either pipe 1 or pipe 2 can be used to run DTC. However, both pipes cannot be used to run DTC.

Website and Support

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

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

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Aug 1, 2014	—	First edition issued
1.10	Dec 26, 2014	—	RX71M is added newly.

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.

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
3. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics product.
5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; and safety equipment etc.
   Renesas Electronics products are neither intended nor authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems, surgical implantations etc.), or may cause serious property damages (nuclear reactor control systems, military equipment etc.). You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application for which it is not intended. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for which the product is not intended by Renesas Electronics.
6. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You should not use Renesas Electronics products or technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. When exporting the Renesas Electronics products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, who distributes, disposes of, or otherwise places the product with a third party, to notify such third party in advance of the contents and conditions set forth in this document, Renesas Electronics assumes no responsibility for any losses incurred by you or third parties as a result of unauthorized use of Renesas Electronics products.
11. This document may not be reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

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.
2880 Scott Boulevard Santa Clara, CA 95050-2554, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada
Tel: +1-905-898-5441, Fax: +1-905-898-3220

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-651-700, Fax: +44-1628-651-804

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-65030, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
7th Floor, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100083, 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 Rd., Putuo District, Shanghai, China
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2886-9318, Fax: +852 2886-9022/9044

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei, 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 906, 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 Korea Co., Ltd.
12F., 234 Teheran-ro, Gangnam-Gu, Seoul, 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141