Renesas USB MCU and USB ASSP

USB Host Human Interface Device Class Driver (HHID)

R01AN0399EJ0210
Rev.2.10
Apr 1, 2013

Introduction

This document describes the use of the USB Host Human Interface Device class driver (referred to here as HHID) with the USB basic firmware and a Renesas USB Device.

Target Device

RX62N Group, RX621 Group, RX63N Group, RX631Group, RX63T Group, R8A66597

The program described here can be used with other RX600 Series microcontrollers that have the same USB module as the above target devices. When using this code in an end product or other application, its operation must be tested and evaluated thoroughly.

This program has been evaluated using Renesas Starter Kit.

Contents

1. Overview

This document describes the use of the Host HID class driver (HHID) and communication port device sample driver with the USB basic firmware with a Renesas USB Device.

1.1 Functions and Features

USB host HID class driver conforms to the USB HID class specification and implements communication with HID devices.
This class driver was developed to be used in combination with Renesas USB Device USB Basic firmware (μITRON version or nonOS version).

1.2 Related Documents

1. Renesas USB Device USB Basic Firmware Application note (Document No.R01AN0512EJ)
2. RX600 Series USB Host Human Interface Devices Class Driver Installation Guide (Document No.R01AN0532EJ)
3. Universal Serial Bus Revision 2.0 specification
   [http://www.usb.org/developers/docs/]
4. USB Class Definitions for Human Interface Devices Version 1.1
5. HID Usage Tables Version 1.1
   [http://www.usb.org/developers/docs/]
6. RX62N Group, RX621 Group User's Manual: Hardware (Document No.R01UH0033EJ)
7. RX63N Group, RX631 Group User's Manual: Hardware (Document No. R01UH0041EJ)
8. RX63T Group User's Manual: Hardware(Document number R01UH0331EJ)
9. R8A66597 Datasheet (Document No. REJ03F0229)
10. RI600/4 User's Manual (Real-time OS for RX Family) (Document No. REJ10J2052）

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

1.3 Terms and Abbreviations

Terms and abbreviations used in this document are listed below.

ANSI	:	ANSI-C File I/O System Calls
APL	:	Application program
ASSP	:	Application Specific Standard Product
cstd	:	Prefix of function and file for Peripheral & Host Common Basic (USB low level) F/W
HCD	:	Host control driver of USB-BASIC-FW
HDCD	:	Host device class driver (device driver and USB class driver)
HEW	:	High-performance Embedded Workshop
HHID	:	Host human interface device
HID	:	Human interface device class
hstd	:	Prefix of function and file for Host Basic (USB low level) F/W
HUBCD	:	Hub class sample driver
MGR	:	Peripheral device state manager of HCD
non-OS	:	USB basic firmware for OS less system
PP	:	Pre-processed definition
RTOS	:	USB basic firmware for uITRON system
RX62N-RSK	:	Renesas Starter Kits for RX62N
RX63N-RSK	:	Renesas Starter Kits for RX63N
RX63T-RSK	:	Renesas Starter Kits for RX63T
R8A66597	:	Renesas Hi-Speed USB2.0 ASSP R8A66597 board (Use in combination with RX62N-RSK.)
Scheduler	:	Used to schedule functions, like a simplified OS.
Scheduler Macro	:	Used to call a scheduler function (non-OS)
SW1/SW2/SW3	:	User switches on the RX62N-RSK / RX63N-RSK (Note1)
Task	:	Processing unit
uITRON, ITRON	:	Industrial The Real-time Operating system Nucleus
USB	:	Universal Serial Bus
USB-BASIC-FW	:	USB basic firmware for Renesas USB MCU and USB ASSP (non-OS/ RTOS)

[Note]
When RX62N-RSK used in conjunction with the R8A66597, SW1 is allocated to the port used by the interruption. Therefore please do not use SW1.

1.4 How To Read This Document

To run the demo, start with the installation guide, listed in chapter 1.2.

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

To get acquainted with the source code, read 3.2 ”File Structure” and note which MCU-specific files you need to copy into directory *HwResourceForUSB*. Also observe which files belong to the application level.

Chapter 4 explains how the default host HID demo application works. You will change this to create your own host user application.

Understand how all code modules are divided into tasks, and that these tasks pass messages to one another. This is so that functions (tasks) can execute in the order determined by a scheduler and not strictly in a predetermined order. This way more important tasks can have priority. Further, tasks are intended to be non-blocking by using a documented callback mechanism. The task mechanism is described in the Basic FW document, listed in 1.2. All HID tasks are listed in 3.3.

2. How to register Class Driver

The user's host class driver will run as a class driver only after "registration".

Consult function usb_hapl_registration() in r_usb_hhid_apl.c to see how a class driver is registered with HCD.

For details, refer to "3.2 How to register Host Class Driver" of a Renesas USB device USB Basic Firmware application note.

3. Software Configuration

3.1 Module Configuration

The HHID comprises the HID class driver and device drivers for mouse and keyboard.

When data is received from the connected USB device, HCD notifies the application. Conversely, when the application issues a request, HCD notifies the USB device.

Figure 3.1 shows the structure of the HHID-related modules. Table 3-1 lists the modules and an overview of each.

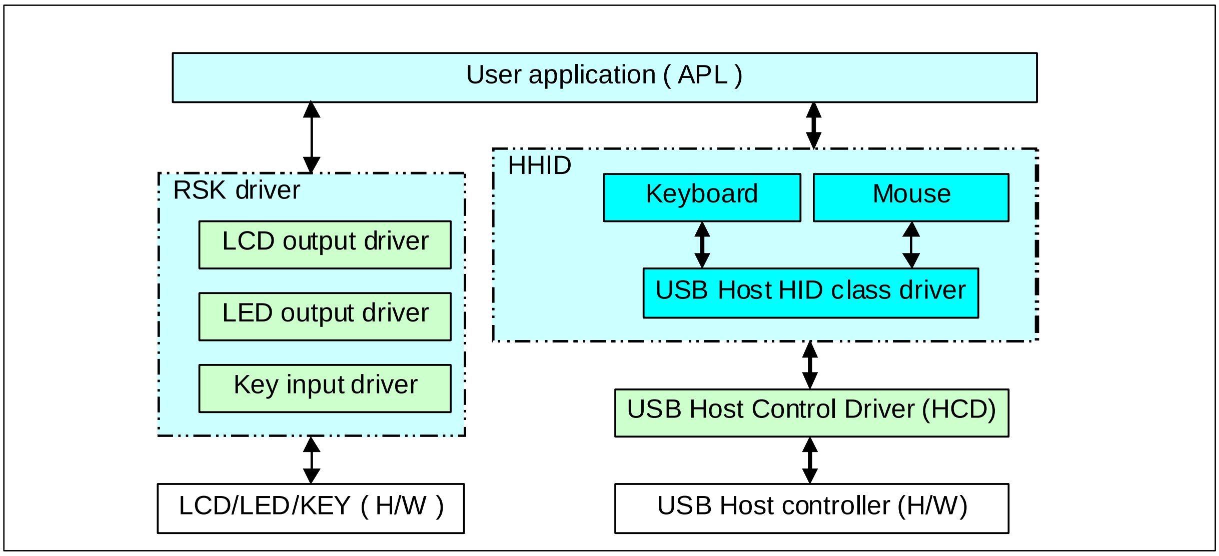

Figure 3.1 Software Configuration

Table 3-1 Module Function Descriptions

Module Name	Description
APL	User application program. Switches initiate communication with HID devices and control suspend/resume. The LCD displays the information received from the HID device.
HHID	The HHID analyzes requests from HID devices. Notifies APL key operation information to the HID host via the HCD.
HCD/MGR	USB host Hardware Control Driver

3.2 File Structure

3.2.1 Folder Structure

The folder structure is shown below. RX device-specific source code is stored in separate hardware resource folders like this: (HwResourceForUSB_***devicename***).

+---WorkSpace

(HHID sample code)

+---HIDFW [*Human I/F Class driver]*

| +---HHID Host HID driver

| +---include Host HID header file

+---SmplMain Sample application

+---HwResourceForUSB *Place files from one of below here.*
File ..\USRCFG\r_usb_usrconfig.h contains user board settings.

+---HwResourceForUSB_RX62N Hardware resource for RX62N/RX621 Group

+---HwResourceForUSB_RX62N_597assp Hardware resource for RX62N+R8A66597

+---HwResourceForUSB_RX63N Hardware resource for RX63N/RX631 Group

+---HwResourceForUSB_RX63T Hardware resource for RX63T Group

(ANSI-C File I/O System Calls) [*open(), close(),read() etc of the USB class driver*]

+--- ANSI

(uITRON)

+---RI600_4 ITRON folder (not included in nonOS version)

(USB-BASIC-FW) [*Common USB code that is used by **all** USB firmware*]

+---USBSTDFW

* 1. Copy the content of folder "HwResourceForUSB_***devicename***" and paste into "HwResourceForUSB" before code compilation.

3.2.2 File Structure

Table 3-2 shows the file structure provided in the HHID. Source files highlighted in aqua may be modified by the user.

Table 3-2 File Structure

Folder	File Name	Description	Notes
HIDFW/HHID	r_usb_hhid_ansi.c	ANSI functions for HHID	
HIDFW/HHID	r_usb_hhid_api.c	API function for HHID	
HIDFW/HHID	r_usb_hhid_driver.c	HHID driver functions	
HIDFW/HHID	r_usb_hhid_defep.c	Pipe Information table	
SmplMain/APL	r_usb_hhid_apl.c	Sample Application program	

3.3 System Resources

3.3.1 System Resource Definitions for RTOS (uITRON) Version

Table 3-3 to Table 3-5 show the uITRON system resources used with the HHID.

Table 3-3 Task information

Task Name	Task Address	Priority	Stacks size	Initial Start	Description
USB_SMP_TSK	usb_cstd_main_task	8	512	ON	Main Task
USB_HCD_TSK	usb_hstd_HcdTask	3	512	OFF	HCD Task
USB_MGR_TSK	usb_hstd_MgrTask	5	512	OFF	MGR Task
USB_HUB_TSK	usb_hhub_Task	4	512	OFF	HUBCD Task
USB_HHID_TSK	usb_hhid_task	6	512	OFF	HHID Task
USB_HHIDSMP_TSK	usb_hhid_MainTask	7	512	OFF	APL Task

Table 3-4 Mailbox information

Mailbox Name	Task Queue	Message Queue	Max Priority	Description
USB_HCD_MBX	FIFO order	FIFO order	1	For HCD
USB_MGR_MBX	FIFO order	FIFO order	1	For MGR
USB_HUB_MBX	FIFO order	FIFO order	1	For HUBCD
USB_ANSI_MBX	FIFO order	FIFO order	1	For ANSI
USB_HHID_MBX	FIFO order	FIFO order	1	For HHID

Table 3-5 Memory pool information

Memory Pool Name	Task Queue	Memory Block		Description
		Number of Blocks	Block Size	
USB_HCD_MPL	FIFO order	10	64	For HCD
USB_MGR_MPL	FIFO order	10	64	For MGR
USB_HUB_MPL	FIFO order	10	64	For HUBCD
USB_ HHID _MPL	FIFO order	10	64	For HHID

3.3.2 System Resource Definitions for Non-OS Version

Table 3-6 to Table 3-8 show the non-OS resources used with the HHID.

Table 3-6 Task information

Function Name	Task ID	Priority	Description
usb_hstd_HcdTask	USB_HCD_TSK	USB_PRI_1	HCD Task
usb_hstd_MgrTask	USB_MGR_TSK	USB_PRI_2	MGR Task
usb_hhub_Task	USB_HUB_TSK	USB_PRI_3	HUBCD Task
usb_hhid_task	USB_HHID_TSK	USB_PRI_3	HHID Task
usb_hhid_MainTask	USB_HHIDSMP_PRI	USB_PRI_4	APL Task

Table 3-7 Mailbox information

Mailbox Name	Using Task ID	Task Queue	Description
USB_HCD_MBX	USB_HCD_TSK	FIFO order	For HCD
USB_MGR_MBX	USB_MGR_TSK	FIFO order	For MGR
USB_HUB_MBX	USB_HUB_TSK	FIFO order	For HUBCD
USB_HHID_MBX	USB_HHID_TSK	FIFO order	For HHID
USB_HHIDSMP_MBX	USB_HHIDSMP_TSK	FIFO order	For APL

Table 3-8 Memory pool information

Memory Pool Name	Task Queue	Memory Block（Note）	Description
USB_HCD_MPL	FIFO order	40byte	For HCD
USB_MGR_MPL	FIFO order	40byte	For MGR
USB_HUB_MPL	FIFO order	40byte	For HUBCD
USB_HHID_MPL	FIFO order	40byte	For HHID
USB_HHIDSMP_MPL	FIFO order	40byte	For APL

[Note]: The maximum number of memory blocks for the entire system is defined in USB_BLKMAX. The default value is 20.

4. Host HID Sample Application Program (APL)

The host HID sample application program was created for use with the RX62N-RSK/RX63N-RSK (RSK herein), and makes use of the three switches (SW2, SW3) and LCD mounted on the RSK.

4.1 Overview

4.1.1 Functions

The main functions of the application are as follows.

1) Display HID reports sent from the USB device on the LCD.

   a) When a USB mouse is connected, and in Mouse mode, the displacement values of the X and Y axes are shown on the LCD. An LED is toggled by pressing the left button.

   b) When a keyboard is connected, and in Keyboard mode, showone character of the key input data from the USB keyboard report.

2) Suspends/resumes the USB device operations.

   a) The USB device is suspended and resumed alternately when SW3 on the RSK is pressed.

   b) Resume is executed when a remote wakeup signal is sent from a remote wakeup device.

4.1.2 Switches

After the USB device is connected, the application executes the following based on user switch input.

1. Data transfer is started by pressing SW2.
2. When SW3 is pushed after starting the data transfer, it shifts to Suspend state.
3. When SW3 is pushed in Suspend state, it cancels Suspend.

Table 4-1 shows switch function on the RSK.

Table 4-1 **RSK Switch Operation**

Switch Name	Switch No.	Operation
Data transfer start	SW2	Request for report reception.
State change	SW3	Change the following USB state. In data reception wait state, go to Suspend state. In Suspend state, go to data reception wait state.

[Note]

Please refer to RSK instructions and MCU Hardware manual regarding the switches on the RSK board and MCU pin connections.

4.2 Displayed Information

The following explains how the LCD display is updated, state transition controls, and other operations.

4.2.1 States

The application displays the USB device connection state and the content of reports received on the LCD.

When a keyboard is connected, the character of the last key pressed on the keyboard is displayed.

When a mouse is connected, the X/Y motion data is displayed . (Values between -128 to 127 are displayed, right justified.

If the content of a received report is NULL (no key press on the keyboard or no XY motion from the mouse), the display on the LCD is not updated.

The following describes the transition of the display state on the LCD.

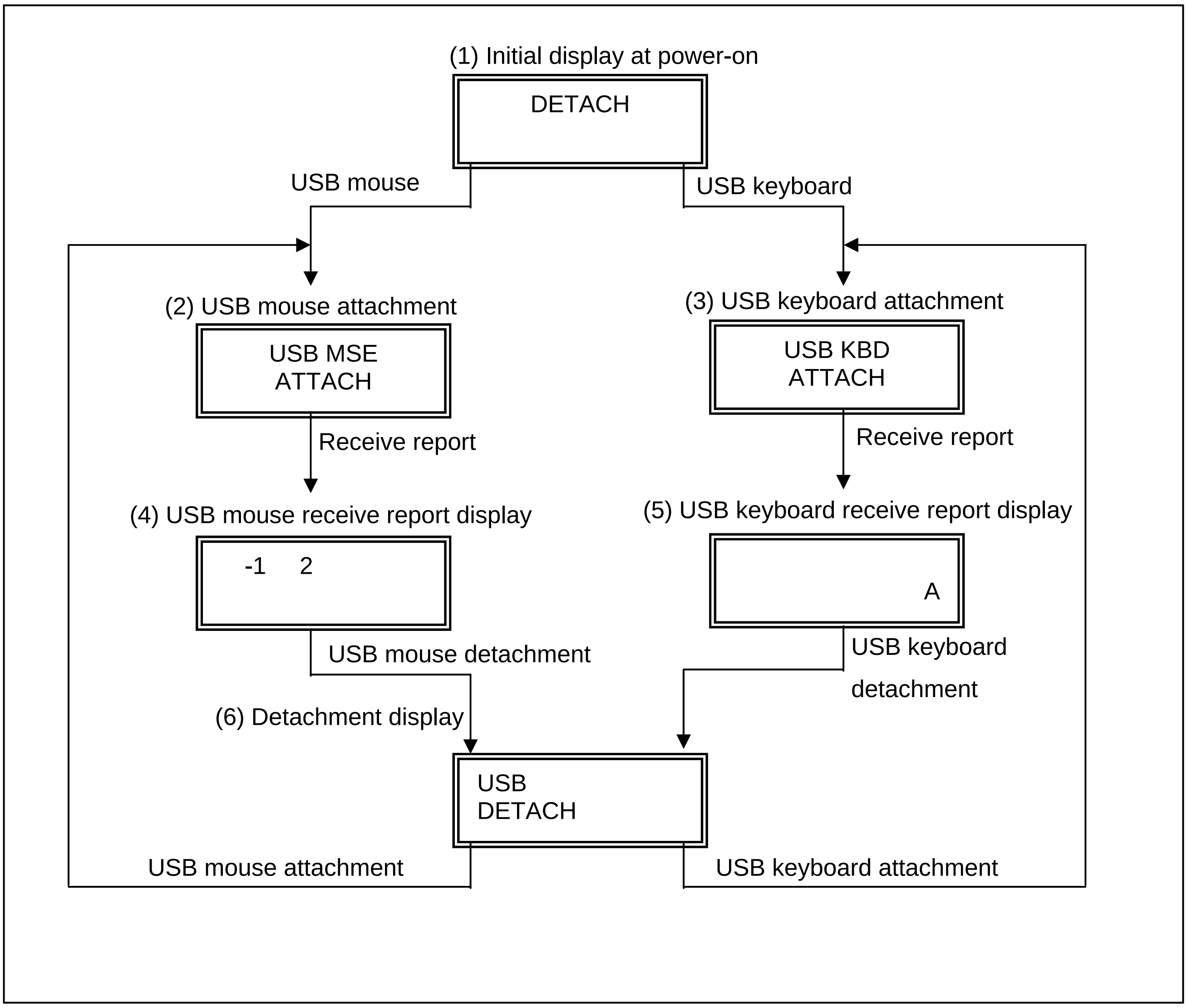

Figure 4.1 The Transition of the Display State on the LCD

4.2.2 State Transitions

Figure 4.2 shows the application state transition. Each block is a program "state".

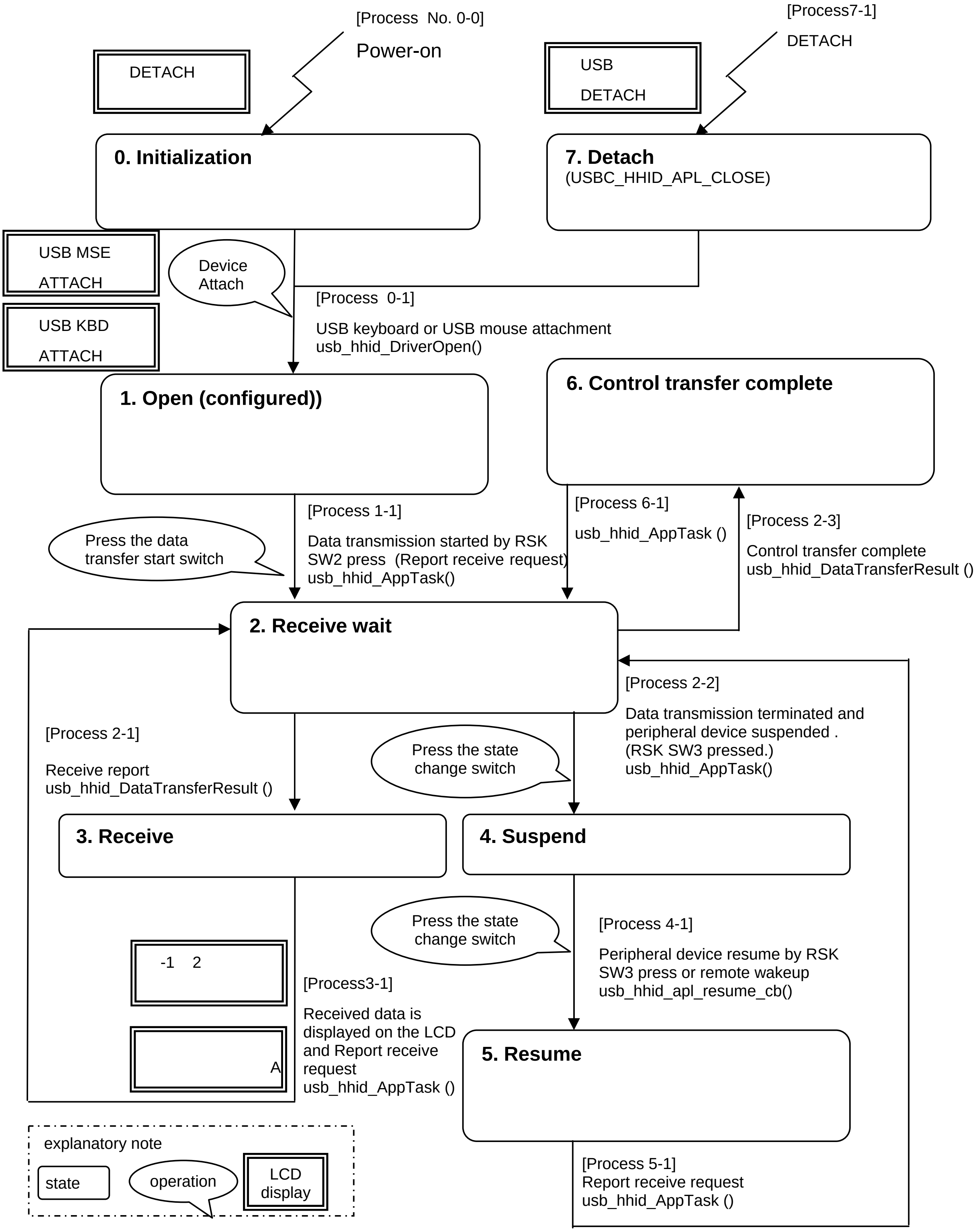

Figure 4.2 Application State Transitions

4.2.3 Operation Descriptions

The following lists application operation with respect to Figure 4.2.

(1). Operating Mode Switching (Process No. 0-1 in Figure 4.2)

The application automatically switches between Keyboard mode and Mouse mode, depending on the bInterfaceProtocol entry in the connected USB device's Configuration descriptor.

This program does not request the report descriptor to USB device.

Table 4.1 Mode Switching Operation

bInterfaceProtocol	Description
0x01	Recognition that a keyboard device is connected
0x02	Recognition that a mouse device is connected
else	Not recognized as an operable HID device connection.

(2). Data Communication Start (Process 1-1 in Figure 4.2**)**

Communication with a USB device is started when "data transfer start switch" is pressed.

(3). Operation during Suspend state (Process 2-2 in Figure 4.2 **)**

Data communication is terminated and the USB device is suspended when "state change switch" is pressed.

(4). Operation during Resume (Process 4-1 in Figure 4.2**)**

USB device is resumed and data communication restarts when "state change switch" is pressed.

(5). Operation during Data Communication (Process 3-1 in Figure 4.2 **)**

Analyzes report received from the peripheral device and displays on LCD.

Table 4.2 Operation During Data Communication

Mode	Description
Keyboard Mode	Display of received key data (key code to ASCII conversion)
Mouse Mode	Display of received coordinate data

4.3 Functions

Table 4.3 lists the application functions for HHID.

Table 4.3 HHID Application Program Functions

	Name	Description
1.	usb_cstd_task_start	Task start processing.
2.	usb_apl_task_switch	Task switching loop for non-OS
3.	usb_hhid_smpl_open	HHID open function (Callback function)
4.	usb_hhid_smpl_close	HHID close function (Callback function)
5.	usb_hapl_task_start	HHID sample start processing.
6.	usb_hhid_task_start	HHID driver start processing.
7.	usb_hapl_registration	HHID driver registration
8.	usb_hhid_prapl_title	Title display.
9.	usb_hhid_smpl_init	Initialize for the global variables of the application
10.	usb_hhid_MainTask	Sample application main processing.
11.	usb_hhid_smpl_data_trans_result	Data transfer complete processing.
12.	usb_hhid_smpl_mse_data	Mouse data reception processing.
13.	usb_hhid_smpl_val_to_str	1-byte numeric data string conversion processing
14.	usb_hhid_smpl_kbd_data	Keyboard data reception processing.
15.	usb_hhid_smpl_resume_cb	Resume complete processing (Callback function)
16.	usb_hhid_smpl_kbd_led_ctl	Keyboard LED ON/OFF setting processing.
17.	usb_hhid_smp_status_set	Sample program mode setting processing.
18.	usb_hhid_smpl_get_hid_descriptor	HID descriptor getting processing.(not used)
19.	usb_hhid_smpl_get_report_descriptor	Report descriptor getting processing.(not used)
20.	usb_hhid_smpl_get_physical_descriptor	Physical descriptor getting processing.(not used)
21.	usb_hhid_smpl_set_report	SET REPORT request transmitting processing. (not used)
22.	usb_hhid_smpl_get_report	GET REPORT request transmitting processing. (not used)
23.	usb_hhid_smpl_set_idle	SET IDLE request transmitting processing.(not used)
24.	usb_hhid_smpl_get_idle	GET IDLE request transmitting processing.(not used)
25.	usb_hhid_smpl_set_protocol	SET PROTOCOL request transmitting processing. (not used)
26.	usb_hhid_smpl_get_protocol	GET PROTOCOL request transmitting processing. (not used)

4.4 Processing Flow Graphs

This following graph shows the application task processing flow (using ANSI-C File I/O Function Calls).

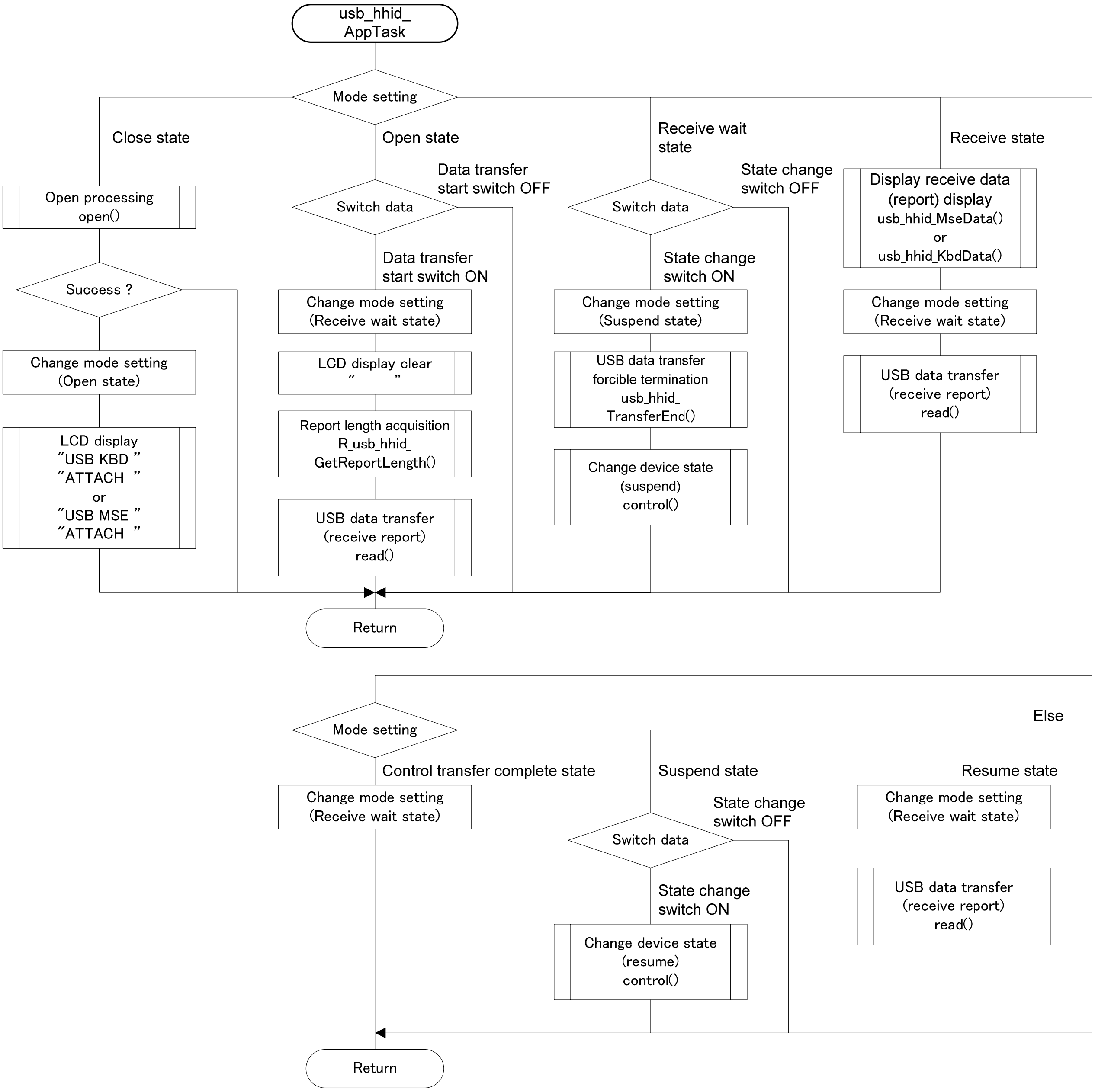
usb_hhid_ AppTask
Mode setting
Close state
Open state
Receive wait state
Receive state
Open processing open()
Success ?
Change mode setting (Open state)
LCD display "USB KBD " "ATTACH " or "USB MSE " "ATTACH "
Data transfer start switch OFF
Switch data
Data transfer start switch ON
Change mode setting (Receive wait state)
LCD display clear " "
Report length acquisition R_usb_hhid_ GetReportLength()
USB data transfer (receive report) read()
State change switch OFF
Switch data
State change switch ON
Change mode setting (Suspend state)
USB data transfer forcible termination usb_hhid_ TransferEnd()
Change device state (suspend) control()
Display receive data (report) display usb_hhid_MseData() or usb_hhid_KbdData()
Change mode setting (Receive wait state)
USB data transfer (receive report) read()
Return
Mode setting
Else
Control transfer complete state
Suspend state
Resume state
Change mode setting (Receive wait state)
State change switch OFF
Switch data
State change switch ON
Change device state (resume) control()
Change mode setting (Receive wait state)
USB data transfer (receive report) read()
Return

Figure 4.3 Application Task Processing Flow Overview (using ANSI-C File I/O Function Calls)

The following shows the application task processing flow overview (non-ANSI-C File I/O Function Calls).

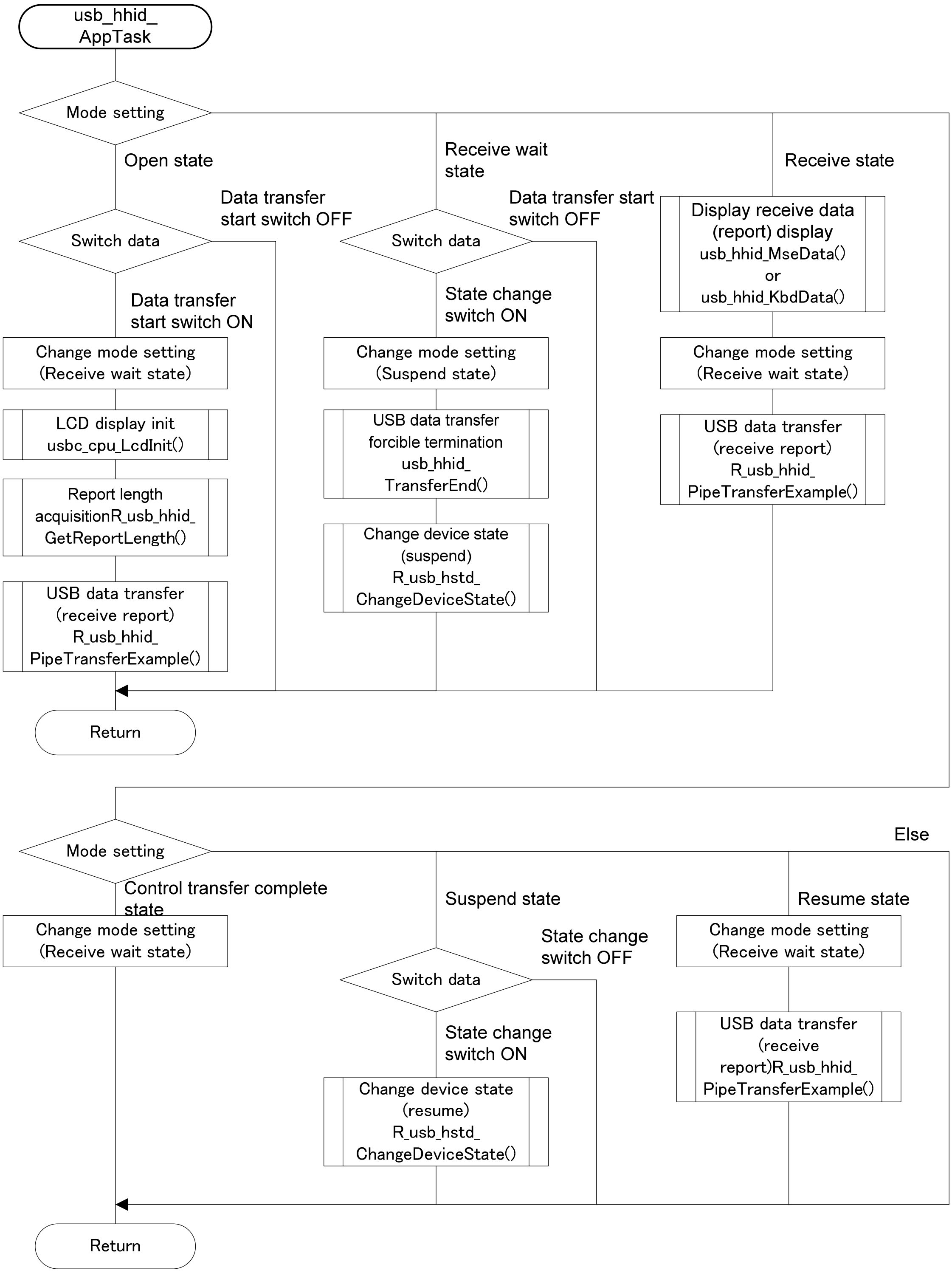

Figure 4.4 Application Task Processing Flow Overview (for non-ANSI-C File I/O Function Calls)

Human Interface Device Class (HID)

4.5 Basic Functions

This software complies with the HID class specification. The main functions of the driver are as follows.

(1) HID device access
(2) Class request notifications to the HID device
(3) Data communication with the HID device

4.6 Class Requests (Host to Device Requests)

Table 4.4 lists the class requests supported by the driver.

Table 4.4 HID Class Requests

Symbol	Request	Code	Description
a	USB_GET_REPORT	0x01	Receives a report from the HID device
b	USB_SET_REPORT	0x09	Sends a report to the HID device
c	USB_GET_IDLE	0x02	Receives a duration (time) from the HID device
d	USB_SET_IDLE	0x0A	Sends a duration (time) to the HID device
e	USB_GET_PROTOCOL	0x03	Reads a protocol from the HID device
f	USB_SET_PROTOCOL	0x0B	Sends a protocol to the HID device
	USB_GET_REPORT_DESCRIPTOR	Standard	Transmits report descriptor
	USB_GET_HID_DESCRIPTOR	Standard	Transmits an HID descriptor

The class request data formats supported in this software are described below.

a). GetReport Request Format

Table 4.5 shows the GetReport request format.
Receives a report from the device in a control transfer.

Table 4.5 GetReport Format

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0xA1	GET_REPORT (0x01)	ReportType & ReportID	Interface	ReportLength	Report

b). SetReport Request Format

Table 4.6 shows the SetReport request format.
Sends report data to the device in a control transfer.

Table 4.6 SetReport Format

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0x21	SET_REPORT (0x09)	ReportType & ReportID	Interface	ReportLength	Report

c). GetIdle Request Format

Table 4.7 shows the GetIdle request format.
Acquires the intarval time of the report notification (interrupt transfer). Idle rate is indicated in 4 ms units.

Table 4.7 GetIdle Format

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0xA1	GET_IDLE (0x02)	0(Zero) & ReportID	Interface	1(one)	Idle rate

d). SetIdle Request Format

Table 4.8 shows the SetIdle request format.
Sets the interval time of the report notification (interrupt transfer). Duration time is indicated in 4 ms units.

Table 4.8 SetIdle Format

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0x21	SET_IDLE (0x0A)	Duration & ReportID	Interface	0(zero)	Not applicable

e). GetProtocol Request Format

Table 4.9 shows the GetProtocol request format.
Acquires current protocol (boot protocol or report protocol) settings.

Table 4.9 GetProtocol Format

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0xA1	GET_PROTOCOL (0x03)	0(Zero)	Interface	1(one)	0(BootProtocol) / 1(ReportProtocol)

f). SetProtocol Request Format

Table 4.10 shows the SetProtocol request format.
Sets protocol (boot protocol or report protocol).

Table 4.10 SetProtocol Format

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0x21	SET_PROTOCOL (0x03)	0(BootProtocol) / 1(ReportProtocol)	Interface	0(zero)	Not applicable

4.6.1 Class Request Structure

The table below shows the structure used by the APIs *control()* and *R_usb_hhid_class_request()* to issue HHID specific requests.

Table 4-2 USB_HHID_CLASS_REQUEST_PARM_t structure

Type	Member name	Description
uint16_t	devadr	Device address
USB_REGADR_t	ipp	USB IP base address
uint16_t	ip	USB IP Number
uint16_t	bRequestCode	Class request code
void*	tranadr	Transfer data buffer
uint32_t	tranlen	Transfer size
uint16_t	duration	Response interval time rate to Interrupt transfer (4ms units)
uint8_t	set_protocol	Protocol value (Boot Protocol(=0)/Report Protocol(=1))
uint8_t*	get_protocol	Protocol value stored address
USB_CB_t	complete	Class request processing end call-back function

4.6.2 HID-Report Format

(1). Receive Report Format

Table 4.11 shows the receive report format used for notifications from the HID device.
Reports are received in interrupt IN transfers or class request GetReport.

Table 4.11 Receive Report Format

Offset	Keyboard Mode	Mouse Mode
Data length	8 Bytes	3 Bytes
0 (Top Byte)	Modifier keys	b0: Button 1 b1: Button 2 b2-7: Reserved
+1	Reserved	X displacement
+2	Keycode 1	Y displacement
+3	Keycode 2	-
+4	Keycode 3	-
+5	Keycode 4	-
+6	Keycode 5	-
+7	Keycode 6	-

(2). Transmit Report Format

Table 4.12 shows the format of the transmit report sent to the HID device.
Reports are sent in the class request SetReport.

Table 4.12 Transmit Report Format

Offset	Keyboard	Mouse
Data length	1 Byte	Not supported
0 (Top Byte)	b0: LED 0 (NumLock) b1: LED 1(CapsLock) b2: LED 2(ScrollLock) b3: LED 3(Compose) b4: LED 4(Kana)	-
+1 ~ +16	-	-

(3). Note

The report format used by HID devices for data communication is based on the report descriptor. This HID driver does not acquire or analyze the report descriptor; rather, the report format is determined by the interface protocol code. User modifications must conform to the HID class specifications.

4.7 Pipe Specification

Table 4.13 shows the pipes used by the driver.
The EP number is determined according to the device endpoint descriptor.

Table 4.13 Pipe Specifications

Number	Transfer Type	Description
PIPE0	Control In/Out	Standard request, class request
PIPE6	Interrupt In	Data transfer from device to host

5. USB Human Interface Device Class Driver (HHID)

5.1 Basic Functions

The HHID driver provides the following basic functions.

(1) Provides data transmit/receive services and a HID device.
(2) Provides HID class request services.

5.2 HHID API Functions

Table 5.1 shows the HHID driver API .

Table 5.1 List of HHID API Functions

	Function	Description
ANSI	open	Establishes the connection with USB device.
	close	Ends the connection with USB device.
	read	Execute USB receive process
	control	Execute process according to control code
Non-ANSI	R_usb_hhid_PipeTransferExample	USB data transfer request to the HCD.
	R_usb_hhid_TransferEnd	USB data transfer termination request to the HCD.
	R_usb_hhid_DeviceInformation	Gets the HID device state information.
	R_usb_hhid_ChangeDeviceState	Changes the device state.
ANSI &Non-ANSI	R_usb_hhid_SetPipeRegistration	Sets the hardware pipe configuration.
	R_usb_hhid_interfaceprotocol	Gets Interface protocol value
	R_usb_hhid_driver_start	HHID driver start processing.
	R_usb_hhid_class_request	Sends the class request
	R_usb_hhid_GetReportLength	Gets the report length.
	R_usb_hhid_DriverRelease	Releases the host HID class.(uITRON only)

[Note]

1. If User selects "USB_ANSIIO_USE_PP" to "Select ANSI Interface" in r_usb_usrconfig.h file, User can call the function described in ANSI field.

   Example (r_usb_usrconfig.h)

   ```
   #define USB_ANSIIO_PP   USB_ANSIIO_USE_PP                  : ANSI I/F used
   ```

2. If User selects " USB_ANSIIO_NOT_USE_PP" to "Select ANSI Interface" in r_usb_usrconfig.h file, User can call the function described in Non-ANSI field.

   Example (r_usb_usrconfig.h)

   ```
   #define USB_ANSIIO_PP      USB_ANSIIO_NOT_USE_PP      : ANSI I/F unused
   ```

3. User can call the function that is described in "ANSI&Non-ANSI" field . These functions are not related to "Select ANSI Interface" setting.

open

Establish connection with a USB device.

Format

int16_t open(int8_t *name, uint16_t mode, uint16_t flg)

Argument

*name	Class Code : USB_CLASS_HHID
mode	Open mode, set to 0 (not used).
flg	Open flag, set to 0 (not used)

Return Value

—	File number (Success: 0x10 -- 0x1F /Failure: -1) As the File Number is required for subsequent communication using read() and write(), the open() function must be called first.

Description

This function will enumerate the connected USB device.
A HW pipe based on the USB information received will be set up, and a connection with the USB device will be established. If enumeration and HW pipe allocation is normal, this function returns (0x10 -- 0x1f) as File number.
If Enumeration and HW pipe allocation fail, (-1)is returned.
After File number is received by the caller, USB device class communications using read() can be performed.
The following Class code is supported.

Class	Class Code	Note
HID	USB_CLASS_HHID	

Note

1. Call this function from the user application program.
2. As File number is required for USB device class communications using read() , the open function must be called before performing the communication.
3. The 2nd argument (mode) and 3rd argument (flag) cannot be used with API : please set both to 0.

Example

```
int16_t  usb_smp_fn;
void usb_apl_task()
{
    :
  usb_smp_fn = open((int8_t *)USB_CLASS_HHID, 0, 0);
  if(usb_smp_fn != -1)
  {
    /* USB Transfer */
  }
    :
}
```

close

End connection with a USB device

Format

int16_t	close(int16_t fileno)

Argument

fileno	File number

Return Value

0	:	Successful
-1	:	Failure

Description

This function ends the connection with the USB device given by File number.

If the operation ends successfully, (0) is returned; if the operation fails, (-1) is returned.

Note

Call this function from the user application program.

Example

```
int16_t usb_smp_fn;
void usb_apl_task()
{
 USB_ER_t err;
     :
 err = close(usb_smp_fn);
 if(err == USB_OK)
 {
    usb_smp_fn = -1;
 }
     :
}
```

read

Receive data from a USB device

Format

int32_t	read(int16_t fileno, uint8_t *buf, int32_t count)

Argument

fileno	File number
*buf	Pointer to data buffer
count	Data transfer size

Return Value

―	Error Code. Always(-1)

Description

This function executes a data receive request for the USB device class specified by File number.

Data is read from the FIFO buffer in the specified data transfer size (3rd argument), and then stored in the data buffer (2nd argument).

When the receive process is complete, the call-back function set in control (USB_CTL_RD_NOTIFY_SET) is called.

The actual read size can be obtained from control (USB_CTL_RD_LENGTH_GET) after the receive process is complete.

Note

1. Call this function from the user application program.
2. Use control (USB_CTL_RD_NOTIFY_SET) to register the call-back function for notification of data transfer complete and then call the API. See "General operation using ANSI API" in the Basic FW manual (listed in 1.2) for more information.
3. This function only executes a data receive request and does not block any processes. Therefore the return value is always -1.

Example

```
int16_t  usb_smp_fn;
void usb_apl_task()
{
    :
  /* Set data receive complete notification call-back */
  control(usb_smp_fn, USB_CTL_RD_NOTIFY_SET, (void*)&usb_ smp_Read_Notify);
  /* receiving data request */
  read(usb_smp_fn, (uint8_t *)buf, (int32_t)size)
  /* receiving request status check */
  err = control(usb_spvendor_bulk_fn, USB_CTL_GET_READ_STATE, (void*)&state);
  if(err != USB_CTL_ERR_PROCESS_COMPLETE)
  {
   /* Error Processing */
  }
    :
}
Processing at the time of the completion of reception
void usb_smp_Read_Notify(USB_UTR_t *ptr, uint16_t data1, uint16_t data2)
{
    :
  /* Receiving data length check */
  err = control(usb_spvendor_bulk_fn, USB_CTL_RD_LENGTH_GET, (void*)&data_len);
  if(err != USB_CTL_ERR_PROCESS_COMPLETE)
  {
   /* Error Processing */
  }
    :
}
```

control

Process according to control code

Format

```
int16_t                control(int16_t fileno, USB_CTRLCODE_t code, void *data)
```

Argument

fileno	File number
code	Control code
*data	Pointer to data (How to use this argument is defferent by Control code. Please refer to Table 5.2 and Example.)

Return Value

0	:	Successful
-1	:	Failure

Description

This function’s processing depends on Control Code.

If an unsupported code is specified, the function sends (-1) as return value.

Table 5.2 Supported Control Codes

Control Code	Description
USB_CTL_USBIP_NUM	Get the USB module number. USB module number is set to the 3rd argument.
USB_CTL_RD_NOTIFY_SET	Register the function called back when a data receive request is completed. Set the call-back function in the 3rd argument.
USB_CTL_RD_LENGTH_GET	Get the data length read from the FIFO buffer when the data is read. The data length is set to the 3rd argument.
USB_CTL_GET_RD_STATE	Get the state when data is read. The state is set to the 3rd argument.
USB_CTL_H_RD_TRANSFER_END	Forcibly end data transfer in pipe relevant to 1st argument(File number).
USB_CTL_H_CHG_DEVICE_STATE	Change state of connected USB device. Set the state value to the 3rd argument.
USB_CTL_H_GET_DEVICE_INFO	Get state of connected USB device. The state is set to the 3rd argument.
USB_CTL_HID_CLASS_REQUEST	Issue a class request for HID, the request is given by

Note

1. Call this function from the user application program.
2. If the user is using the ANSI method and specifies “USB_CTL_HID_CLASS_REQUEST” as Control code (2nd argument), the user can issue the following class requests. Please assign the definition of the Class Request to the “bRequestCode” member.

Class Request	Definition Value
Get_Descriptor(HID)	USB_HID_GET_HID_DESCRIPTOR
Get_Descriptor(Report)	USB_HID_GET_REPORT_DESCRIPTOR
Get_Descriptor(Physical)	USB_HID_GET_PHYSICAL_DESCRIPTOR
Set_Report	USB_HID_SET_REPORT
Get_Report	USB_HID_GET_REPORT
Set_Idle	USB_HID_SET_IDLE
Get_Idle	USB_HID_GET_IDLE
Set_Protocol	USB_HID_SET_PROTOCOL
Get_Protocol	USB_HID_GET_PROTOCOL

Example

```
＜USB_CTL_USBIP_NUM＞
int16_t  usb_smp_fn;
void usb_apl_task(USB_UTR_t *ptr)
{
  int16_t num;
    :
  /* Confirmation USBIP Number */
  control(usb_smp_fn, USB_CTL_USBIP_NUM, (void*) &num);
    :
}
```

```
＜USB_CTL_H_RD_TRANSFER_END＞
int16_t  usb_smp_fn;
void usb_apl_task(USB_UTR_t *ptr)
{
  USB_CTL_PARAMETER_t smp_parameter;
    :
  smp_parameter.transfer_end.status = USB_DATA_STOP;
  /* Forcibly ends data reception */
  control(usb_smp_fn, USB_CTL_H_RD_TRANSFER_END, (void)&smp_parameter);
    :
}
```

```
＜USB_CTL_H_CHG_DEVICE_STATE＞
int16_t  usb_smp_fn;
void usb_apl_task(USB_UTR_t *ptr)
{
  USB_CTL_PARAMETER_t smp_parameter;
    :
  smp_parameter.dev_info.complete = ptr.complete;   /* Callback function */
  smp_parameter.dev_info.msginfo = USB_DO_STALL;
  /* Changing USB device information */
  control(usb_smp_fn, USB_CTL_H_CHG_DEVICE_STATE, (void)&smp_parameter);
    :
}
＜USB_CTL_H_GET_DEVICE_INFO＞
int16_t  usb_smp_fn;
void usb_apl_task(USB_UTR_t *ptr)
{
  USB_CTL_PARAMETER_t smp_parameter;
    :
  smp_parameter.device_information.tbl = &smp_tbl;
  /* Getting USB device information */
  control(usb_smp_fn, USB_CTL_H_GET_DEVICE_INFO, (void)&smp_parameter);
    :
}

＜USB_CTL_HID_CLASS_REQUEST＞
void usb_apl_task(USB_UTR_t *ptr)
{
  USB_HHID_CLASS_REQUEST_PARM_t   class_req;

  class_req.bRequestCode = USB_HID_GET_HID_DESCRIPTOR; /* Class Reqeust */
  class_req.devadr = devadr; /* Device address of HID device */
  class_req.ip = ptr->ip;
  class_req.ipp = ptr->ipp;
  /* Pointer to the buffer that the class request*/
  class_req.tranadr = p_data;  class_req.complete = complete;

  /* HID Class Request */
  control(usb_smp_fn, USB_CTL_HID_CLASS_REQUEST, (void*)&class_req );
}

＜USB_CTL_RD_NOTIFY_SET＞
＜USB_CTL_GET_RD_STATE＞
＜USB_CTL_RD_LENGTH_GET＞
  Please refer to example of "read" function.
```

R_usb_hhid_PipeTransferExample

USB data transfer request

Format

USB_ER_t	R_usb_hhid_TransferExample(USB_UTR_t *ptr, uint8_t *table, uint32_t size, USB_CB_t complete)

Argument

*ptr	Pointer to USB Transfer structure
*table	Pointer to the data buffer area
size	Read data size
complete	Call-back function

Return Value

［non-OS］

USB_E_OK	Success
USB_E_ERROR	Failure

［RTOS］

－	Error Code. Please refer to RI600/4 User's manual for RX family Real-time OS.

Description

This function requests a data transfer to the USB device.

When data transfer ends (specified data size reached, short packet received, error occurred), the call-back function is called.

Information on remaining transmit/receive data length, status, error count and transfer end is available in the parameter of the call-back function.

Note

1. Call this function from the user application program or class driver.
2. Please set the following member of USB_UTR_t structure.

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

3. This function is not necessary when using an ANSI IO API.

Example

```
USB_ER_t usb_smp_task(void)
{
  USB_UTR_t usbip;

  usbip.ip = USB_HOST_USBIP_NUM;
  usbip.ipp = R_usb_cstd_GetUsbIpAdr( usbip.ip );
                          :
                          :
  R_usb_hhid_TransferExample(&usbip,buf,size,(USB_CB_t)usb_data_received);

}

/* Callback function */
void usb_data_received(USB_UTR_t *mess)
{
           :
}
```

R_usb_hhid_TransferEnd

USB data transfer termination request

Format

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

Argument

*ptr	Pointer to USB Transfer structure
pipe	Pipe No.
status	USB communication status

Return Value

〔non-OS〕

USB_E_OK	Success.
USB_E_ERROR	Failure

〔RTOS〕

―	Error Code. Please refer to RI600/4 User's manual for RX family Real-time OS.

Description

This function forces data transfer via the pipes to end.

The function executes a data transfer forced end request to the HCD. After receiving the request, the HCD 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_hhid_PipeTransferExample) at the time 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 to the argument (ptr) of this callback function

Note

1. Call this function from the user application program or class driver.
2. Please set the following member of USB_UTR_t structure.

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

3. This function is not necessary when using an ANSI IO API.

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_hhid_TransferEnd(ptr, pipe, status);

   return err;
     :
}
```

R_usb_hhid_DeviceInformation

Get the HID device information

Format

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

Argument

*ptr	Pointer to USB Transfer structure
devaddr	USB device address
*tbl	Pointer to the table address for device information storing

Return Value

—	—

Description

The information on the device connected to the USB port is acquired.

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

［0］ Root port number to which device is connected

［1］ Device state

［2］ Configuration number

［3］ Interface class code 1

［4］ Connection speed

［5］ --

［6］ --

［7］ --

［8］ Status of rootport0

［9］ Status of rootport1

Note

1. Call this function from the user application program or class driver.
2. This function is not necessary when using an ANSI IO API.
3. Please set the following member of USB_UTR_t structure.

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

4. As USB Host Human Interface Device Class Driver (HHID) 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)
{
  USB_UTR_t usbip;
  uint16_t      tbl[10];
    :
  usbip.ip = USB_HOST_USBIP_NUM;     /* Setting USB IP No */
  /* Confirm the device information */
  R_usb_hhid_DeviceInformation(ptr, devaddr, &tbl);
    :
}
```

R_usb_hhid_ChangeDeviceState

Changes device state

Format

void	R_usb_hhid_ChangeDeviceState(USB_UTR_t *ptr, uint16_t msginfo)

Argument

*ptr	Pointer to USB Transfer structure
msginfo	USB communication status

Return Value

—	—

Description

This function changes the device state.

The following values are set to msginfo and change of the USB device State is required of HCD by calling this function.

msginfo	Description
USB_DO_GLOBAL_SUSPEND	Request to change to suspend state
USB_DO_GLOBAL_RESUME	Request to execute resume signal

Note

1. Call this function from the user application program or class driver.
2. Please set the following member of USB_UTR_t structure.

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

3. This function is not necessary when using an ANSI IO API.

Example

```
void  usb_smp_task( void )
{
  USB_UTR_t usbip;

  usbip.ip = USB_HOST_USBIP_NUM;
  usbip.ipp = R_usb_cstd_GetUsbIpAdr( USB_HOST_USBIP_NUM );
    :
  /* Change the device state request */
  R_usb_hhid_ChangeDeviceState(ptr, USB_DO_GLOBAL_SUSPEND);
    :
}
```

R_usb_hhid_SetPipeRegistration

Set USB hardware pipe configuration

Format

void	R_usb_hhid_SetPipeRegistration(USB_UTR_t *ptr, uint16_t devadr)

Argument

*ptr	Pointer to USB Transfer structure
devadr	USB device address

Return Valaue

—	—

Description

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

Note

1. Call this function from the user application program during initialization.
2. Please set the following member of USB_UTR_t structure.

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

3. This function is not necessary when using an ANSI IO API.

Example

```
void  usb_smp_task( void )
{
    :
  R_usb_hhid_SetPipeRegistration (ptr, devadr);
    :
}
```

R_usb_hhid_get_interfaceprotocol

Get interface protocol value

Format

uint8_t	R_usb_hhid_get_interfaceprotocol(void)

Argument

—	—

Return Value

—	Protocol code of USB device（bInterfaceProtocol）

Description

This function gets the interface protocol value of the connected USB device.

Note

1. Call this function from the user application program or class driver.
2. bInterfaceProtocol is included in Interface Descriptor.

Example

```
void  usb_smp_task( void )
{
  uint8_t   protocol;
    :
  /* Gets the interface protocol value */
  protocol = R_usb_hhid_get_interfaceprotocol();
    :
}
```

R_usb_hhid_driver_start

HHID driver start

Format

void	R_usb_hhid_driver_start(USB_UTR_t *ptr)

Argument

*ptr	Pointer to USB Transfer structure

Return Value

—	—

Description

［non-OS］

This function sets the priority of HHID driver task.

The sent and received of message are enable by the priority is set.

［RTOS］

This function starts HHID driver task.

Note

1. Call this function from the user application program during initialization.
2. Please set the following member of USB_UTR_t structure.

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

Example

```
void usb_hstd_task_start( void )
{
  USB_UTR_t  *ptr;
    :
  ptr->ip = USB_HOST_USBIP_NUM;   /* USB IP No */
  ptr->ipp = R_usb_cstd_GetUsbIpAdr( ptr->ip );  /* USB IP base address */
    :
  R_usb_hhid_driver_start( ptr );    /* Host Class Driver Task Start Setting */
  usb_hstd_usbdriver_start( ptr );   /* Host USB Driver Start Setting */
  usb_hapl_registration( ptr );      /* Host Application Registration */
  usb_hapl_task_start( ptr );        /* Host Application Task Start Setting */
    :
}
```

R_usb_hhid_class_request

Send HID class request

Format

USB_ER_t R_usb_hhid_class_request(void *pram)

Argument

*pram HID class request structure. See 4.6.1 .

Return Value

［non-OS］

USB_E_OK	Success.
USB_E_ERROR	Failure

［RTOS］

―	Error Code. Please refer to RI600/4 User's manual for RX family Real-time OS.

Description

This function request HID class request issue to HID driver.

Note

1. Call this function from the user application program or class driver. Please refer to "Example".
2. This function is not necessary when using an ANSI IO API. In that case just call the Control API with "USB_CTL_HID_CLASS_REQUEST" as 2nd argument, and do not use this function.
3. If the ANSI IO method is *not* used, the class requests listed below can be called using this API. Please assign the desired Request Code to "bRequestCode" member before calling.

Class Request	Definition Value
Get_Descriptor(HID)	USB_HID_GET_HID_DESCRIPTOR
Get_Descriptor(Report)	USB_HID_GET_REPORT_DESCRIPTOR
Get_Descriptor(Physical)	USB_HID_GET_PHYSICAL_DESCRIPTOR
Set_Report	USB_HID_SET_REPORT
Get_Report	USB_HID_GET_REPORT
Set_Idle	USB_HID_SET_IDLE
Get_Idle	USB_HID_GET_IDLE
Set_Protocol	USB_HID_SET_PROTOCOL
Get_Protocol	USB_HID_GET_PROTOCOL

Example

```
void usb_hhid_smpl_set_report(USB_UTR_t *ptr, uint16_t devadr, uint8_t *p_data,
uint16_t length, USB_CB_t complete)
{
  USB_HHID_CLASS_REQUEST_PARM_t   class_req;

  /* SET_REPORT */
  class_req.bRequestCode = USB_HID_SET_REPORT;

  class_req.devadr = devadr;
  class_req.ip = ptr->ip;
  class_req.ipp = ptr->ipp;
  class_req.tranadr = p_data;
  class_req.tranlen = length;
  class_req.complete = complete;

    R_usb_hhid_class_request((void*)&class_req);

}
```

R_usb_hhid_GetReportLength

Gets HID Report length

Format

uint16_t	R_usb_hhid_GetReportLength(void)

Argument

—	—

Return Value

—	Max packet size

Description

This function gets the max packet size of the connected USB device.

Note

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

Example

```
void usb_smp_task( void )
{
  uint16_t   usb_smp_report_length;
    :
  usb_smp_report_length = R_usb_hhid_GetReportLength();
    :
}
```

R_usb_hhid_DriverRelease

Release Host HID class (uITRON only)

Format

void	R_usb_hhid_DriverRelease(USB_UTR_t *ptr)

Argument

*ptr	Pointer to USB Transfer structure

Return Value

—	—

Description

Release the registered HHID.

Note

1. When the registered HHID is unnecessary, please call this function in the user application program or class driver.
2. Please set the following member of USB_UTR_t structure.

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

Example

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

5.2.1 HHID Task Callback Functions

This section describes the HHID task callback functions.

Table 5.3 usb_hhid_DriverOpen()

Name	HHID Open Function		
Call format	usb_hhid_DriverOpen		
Arguments	uint16_t	data1, data2	Unused
Return values	void	-	-
Description	Callback function that is called from USB driver by a Set_Configuration request being correctly executed. Displays information on the LCD according to the application mode. Performs the hardware registration for the pipe used by the HHID. For the RTOS version, this function creates and runs the HHID tasks and creates the HHID mailbox.		
Notes			

Table 5.4 usb_hhid_DriverClose()

Name	HHID Close Function		
Call format	usb_hhid_DriverClose		
Arguments	uint16_t	data1, data2	Unused
Return values	void	-	-
Description	Callback function that is called from USB driver by a detach detection. Displays the fact that an detach has occurred on the LCD and clears application mode.		
Notes			

Table 5.5 usb_hhid_ClassCheck()

Name	Connected Device Verification at Enumeration (Initialization)		
Call format	usb_hhid_ClassCheck		
Arguments	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
Return values	void	-	-
Description	Performs processing called from the HHID class check. Since information for the connected device has been reported, this function saves that information in the global area. Also initializes pipe information table and deletes mailbox.		
Notes			

Table 5.6 usb_hhid_TransferResult()

Name	Report Data Transfer Complete to HHID Task		
Call format	usb_hhid_TransferResult		
Arguments	USBC_UTR_t *	mess	Processing results
Return values	void	-	-
Description	Callback function that is called on the completion of a data transfer. Transfers the processing result passed as its argument to the HHID task mailbox.		
Notes			

Table 5.7 usb_hhid_ClassTransResult()

Name	Class Request Communication Complete Callback		
Call format	usb_hhid_ClassTransResult		
Arguments	USBC_UTR_t *	mess	Processing results
Return values	void	-	-
Description	This is the callback function called when a class request communication is completed. It transfers the processing results passed in an argument to the class request processing mailbox.		
Notes	This function is for ITRON operations.		

5.3 HHID Task Description

This task receives messages addressed to the HHID and performs processing according to the type of message.

Table 5.8 Processing according to Received HHID Message Type shows the processing according to the message type.

Table 5.8 Processing according to Received HHID Message Type

Enumeration Sequence Variable (usb_ghhid_EnumerationSequence)	Processing	Message Source

During enumeration
(other than USBC_HHID_ENUM_COMPLETE)
	Gets the string descriptor and sets the pipe according the enumeration sequence.	usb_hhid_ClassCheck()
		：Class check call-back function
		usb_hhid_TransferResult()
		：Standard Request GetDescriptor communication complete call-back function

After enumeration completion (other than USBC_HHID_ENUM_COMPLETE)	Notifies the application of a data transfer completion.	usb_hhid_TransferResult() ：Interrupt IN transfer complete call-back function

*HHID messages are not received in states other than the configuration state. Operations during enumeration are only available in non-OS operations; operations in the RTOS version must wait until enumeration completes.

6. Target Peripheral List (TPL)

The user needs to make the target peripheral list (TPL) that the HHID driver will enumerate. Refer to the TPL (usb_gapl_devicetpl []) that is defined in the user application program (r_usb_hhid_apl.c).

For details, refer to “5.6 Target Peripheral List” in USB Basic Firmware Application Note.

7. How to register to uITRON and Non-OS Versions

For details, please refer to "9.1 How to register in non-OS " and "9.2 How to register in RTOS " of the Renesas USB device USB Basic Firmware application note.

8. Limitations

The following limitations apply to the HHID.

1. The HID driver must analyze the report descriptor to determine the report format (This HID driver determines the report format from the interface protocol alone.)
2. Low Speed operation is not supported by Renesas USB devices in the RX62N, RX63N and RX63T groups.
3. The number of devices that can be connected is one in this F/W.

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	Mar.09.11	—	First edition issued
2.00	Mar.28.12	—	First edition issued for V.2.00
2.01	Feb.01.13	—	Description mistake and the reference error is fixed
2.10	Apr.01.13	—	First Release for V.2.10 Add Target Device RX63T.Add the information on RX63T

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 accord 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.

RENESAS

Renesas Electronics Corporation

SALES OFFICES

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 204, 205, AZIA Center, No.1233 Lujiazui Ring Rd., Pudong District, Shanghai 200120, China
Tel: +86-21-5877-1818, Fax: +86-21-6887-7858 / -7898

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.
11F., Samik Lavied' or Bldg., 720-2 Yeoksam-Dong, Kangnam-Ku, Seoul 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 2.2