USB Device RNDIS Class Driver User Guide

Version 1.10

For use with USBD RNDIS Class Driver Version 2.1 and above

Date: 16-Jun-2017 13:32

Table of Contents

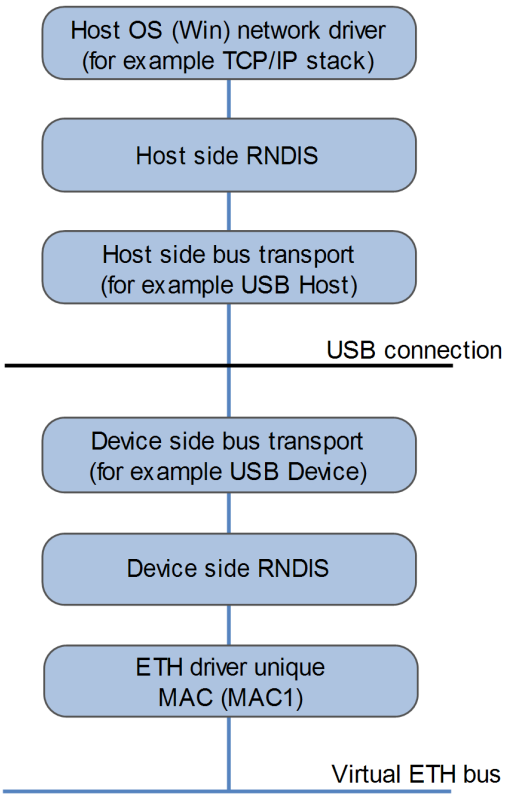
1 System Overview

1.1 Introduction

This guide is for those who want to implement an embedded USB device RNDIS class driver. The Remote Network Driver Interface Standard (RNDIS) is used to provide a virtual Ethernet link over USB. It is a Microsoft proprietary protocol.

The **usbd_cd_rndis** package is a function device implementation of this class. This allows a device to connect to a host system and appear to it as one or more locally connected virtual Ethernet ports. This device class driver provides a network driver interface that conforms to HCC's Network Driver interface standard, described in the *HCC Network Driver User Guide*.

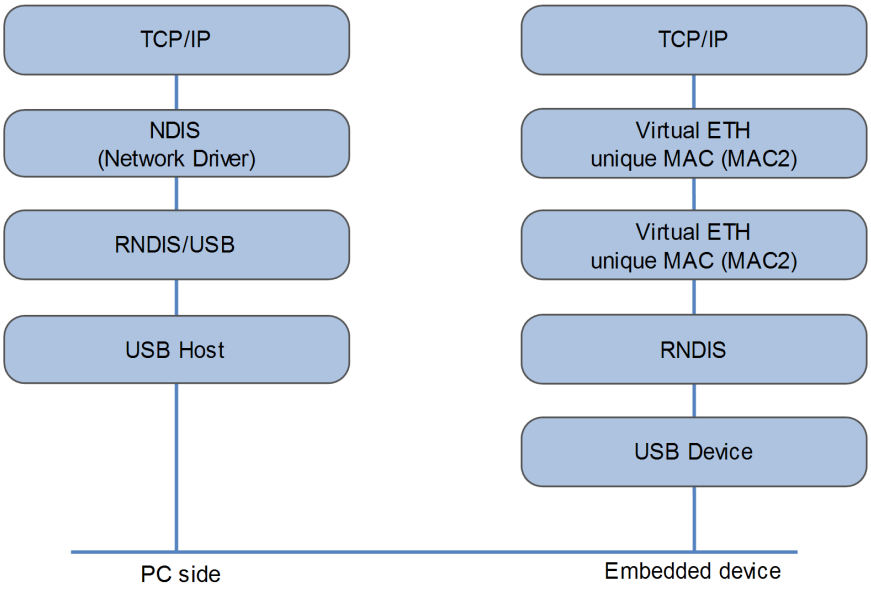
The system structure is shown in the diagram below:



The USB device port creates a virtual Ethernet port on the embedded device. This connects to a physical Ethernet port and thus to a network. If the embedded device wants to be a device on that network then we provide a virtual network on the embedded device - so that the network driver logically talks to another virtual Ethernet port inside the device - and a network stack can then be built on that interface.

In this virtual network configuration the system will probably need a simple DHCP server to provide an IP address to the remote network Ethernet port of the PC that is in the HCC system.

This diagram shows access to the TCP/IP stack over USB/RNDIS:



This class driver is effectively a library. It provides a set of function calls that an application can use to send and receive data through the serial port. The **usbd_rndis_init()** function registers the class driver with the Embedded USB Device (EUSBD) base system and this call sets up callbacks for the base system to use.

Note: This module is part of HCC's EUSBD system, as described in the *HCC Embedded USB Device Base System User Guide*. This module communicates with the EUSBD base system through the EUSBD Device Interface, as described in the above manual.

1.2 Feature Check

The main features of the class driver are the following:

- Conforms to the HCC Advanced Embedded Framework.
- Conforms to HCC's Network Driver interface standard, described in the *HCC Network Driver User's Guide*.
- Designed for integration with both RTOS and non-RTOS based systems.
- Supports all devices that conform to the RNDIS specification.
- Compatible with sample device files produced by using HCC's *USB Device Descriptor Generator*.
- Allows the user to specify a callback for state change events.

1.3 Packages and Documents

Packages

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

Package	Description
hcc_base_doc	This contains the two guides that will help you get started.
usbd_base	The USB device base package. This is the framework used by USB class drivers to communicate over USB using a specific USB device controller package.
nw_drv_base	The base HCC network driver package.
usbd_cd_rndis	The USB device RNDIS class driver package described by this document.

Documents

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

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

HCC Firmware Quick Start Guide

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

HCC Source Tree Guide

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

HCC Embedded USB Device Base System User Guide

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

HCC Network Driver User Guide

This document describes how to implement an HCC network driver.

HCC USB Device RNDIS Class Driver User Guide

This is this document.

HCC USB Device Descriptor Generator User Guide

This document describes the tool that creates USB descriptor files for inclusion in a project that uses the EUSBD stack.

1.4 Change History

This section describes past changes to this manual.

- To view or download earlier manuals, see Archive: Embedded USB Device RNDIS Class Driver User Guide.
- For the history of changes made to the package code itself, see History: usbd_cd_rndis.

The current version of this manual is 1.10. The full list of versions is as follows:

Manual version	Date	Software version	Reason for change
1.10	2017-06-16	2.1	New *Change History* format. Function group tables in API section. New configuration option.
1.00	2015-05-08	1.16	First release.

2 Source File List

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

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

2.1 API Header File

The file **src/api/api_usbd_rndis.h** is the only file that should be included by an application using this module. For details of the API functions, see Application Programming Interface.

2.2 Configuration File

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

2.3 Source Code

The file **src/usb-device/class-drivers/rndis/usbd_rndis.c** contains the main code for USB device RNDIS class drivers. **This file should only be modified by HCC**.

2.4 Version File

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

3 Configuration Options

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

USBD_RNDIS_MAX_TOTAL_SIZE

The maximum total packet size. The default is 1514.

USBD_RNDIS_MAX_PKT_SIZE

The maximum packet size. The default is USBD_RNDIS_MAX_TOTAL_SIZE - 14.

USBD_RNDIS_VENDOR_DESCRIPTION

The vendor description. The default is "HCC-Embedded V1.0".

USBD_RNDIS_VENDOR_ID

The vendor ID. The default is 0xC1CA.

USBD_RNDIS_ACT_AS_NWDRIVER

Keep the default of 1 if you want RNDIS to act as a network driver. Here the class driver acts as a network driver and it can be interfaced to a network interface. This is typically required if the class driver is used with HCC's TCP/IP stack.

Set this to 0 to interface the network driver to RNDIS. Here the class driver acts as a network interface: a network driver can be interfaced to it. This is typically required if the class driver needs to be used standalone or with a non HCC TCP/IP stack.

USBD_RNDIS_COM_TASK_STACK_SIZE

The size of the COM task stack. The default is 1024.

> **Note:** The following three options only apply if **USBD_RNDIS_ACT_AS_NWDRIVER** is set to 0.

USBD_RNDIS_MAX_RX_USB_BUF_NUM

The maximum number of buffers for receiving from USB. The default is 5.

USBD_RNDIS_RX_TASK_STACK_SIZE

The size of the receive task stack. The default is 1024.

USBD_RNDIS_TX_TASK_STACK_SIZE

The size of the transfer task stack. The default is 1024.

USBD_RNDIS_DEFAULT_LINK_SPEED

The default link speed. The default is 10000000 (10 Mbps).

USBD_RNDIS_DEFAULT_NETWORK_ADDR

The default network address. The default is `{ 0x00, 0xA1, 0x92, 0x00, 0x92, 0x07 }`.

4 Network Address Assignment

When you connect a network stack to a PC runnning RNDIS, you must assign network addresses to each end of the connection. You can do this statically or you can add HCC's DHCP Server module to the device to assign a network address to the host PC.

5 Application Programming Interface

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

5.1 Module Management

The functions are the following:

Function	Description
usbd_rndis_init()	Initializes the module and allocates the required resources.
usbd_rndis_start()	Starts the module.
usbd_rndis_stop()	Stops the module.
usbd_rndis_delete()	Deletes the module and releases the resources it used.

usbd_rndis_init

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

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

Format

```
int usbd_rndis_init (
    t_nwdriver_init   p_nwd_drv_init,
    uint32_t          param )
```

Arguments

Parameter	Description	Type
p_nwd_drv_init	The NWDRIVER initialization function.	t_nwdriver_init
param	A driver-specific parameter (not used currently).	uint32_t

Return Values

Return value	Description
USBD_RNDIS_SUCCESS	Successful execution.
USBD_RNDIS_ERR_RESOURCE	Resource (mutex, event, or task) allocation error.

usbd_rndis_start

Use this function to start the class driver.

> **Note:** You must call **usbd_rndis_init()** before this to initialize the module.

Format

```
int usbd_rndis_start ( void )
```

Arguments

Parameter
None.

Return Values

Return value	Description
USBD_RNDIS_SUCCESS	Successful execution.
USBD_RNDIS_ERROR	Operation failed.

usbd_rndis_stop

Use this function to stop the class driver.

Format

```
int usbd_rndis_stop ( void )
```

Arguments

Parameter
None.

Return Values

Return value	Description
USBD_RNDIS_SUCCESS	Successful execution.
USBD_RNDIS_ERROR	Operation failed.

usbd_rndis_delete

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

Format

```
int usbd_rndis_delete ( void )
```

Arguments

Parameter
None.

Return Values

Return value	Description
USBD_RNDIS_SUCCESS	Successful execution.
USBD_RNDIS_ERROR	Operation failed.

usbd_rndis_init_ethdrv

Use this function to get the transfer status of the current transfer in the requested direction.

> **Note:** This is only to be used if USBD_RNDIS_ACT_AS_NWDRIVER is non-zero.

Format

```
int usbd_rndis_init_ethdrv ( uint8_t comm_idx )
```

Arguments

Parameter	Description	Type
comm_idx	The communication index to be used by the network driver to address the RNDIS communication class driver.	uint8_t

Return Values

Return value	Description
USBD_SUCCESS	Successful execution.
USBD_ERROR	Operation failed.

5.2 Device Management

The functions are the following:

Function	Description
usbd_rndis_set_mac_addr()	Sets a new RNDIS MAC address.
usbd_rndis_register_cb()	Registers a state change callback function.

usbd_rndis_set_mac_addr

Use this function to set a new RNDIS MAC address.

> **Note:** This function can only be called if RNDIS is in the disconnected state.

Format

```
int usbd_rndis_set_mac_addr ( const uint8_t * const p_mac_address )
```

Arguments

Parameter	Description	Type
p_mac_address	The MAC address.	uint8_t *

Return Values

Return value	Description
USBD_RNDIS_SUCCESS	Successful execution.
USBD_RNDIS_ERROR	Operation failed.

usbd_rndis_register_cb

Use this function to register a state change callback function.

> **Note:** It is the user's responsibility to provide any callback functions the application requires. Providing such functions is optional.

Format

```
int usbd_rndis_register_cb ( t_usbd_rndis_state_change_cb sc_cb )
```

Arguments

Parameter	Description	Type
sc_cb	The state change callback function.	t_usbd_rndis_state_change_cb

Return Values

Return value	Description
USBD_RNDIS_SUCCESS	Successful execution.
USBD_ERROR	Operation failed.

5.3 Error Codes

If a function executes successfully, it returns with USBD_RNDIS_SUCCESS, a value of zero. The following table shows the meaning of the error codes.

Return Code	Value	Description
USBD_RNDIS_SUCCESS	0	Successful execution.
USBD_RNDIS_ERR_RESOURCE	1	Resource (mutex, event, or task) allocation error.
USBD_RNDIS_ERROR	2	Operation failed.

5.4 Types and Definitions

t_usbd_rndis_state_change_cb

The **t_usbd_rndis_state_change_cb** definition specifies the format of the callback function that can be called when a state change occurs on the control channel.

Format

```
typedef void ( *t_usbd_rndis_state_change_cb )(
    uint8_t   uid,
    uint8_t   state )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	uint8_t
state	The device state.	uint8_t

Device States

The posssible device states are as follows:

Return Code	Value	Description
USBD_RNDIS_ST_DISCONNECTED	0	No device is connected.
USBD_RNDIS_ST_CONNECTED	1	A device is connected.

6 Integration

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

6.1 OS Abstraction Layer

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

The RNDIS module uses the following OAL components:

Resource	If not used as network interface	If used as network interface
Tasks	1	3
Mutexes	0	2
Events	2	6

6.2 PSP Porting

The Platform Support Package (PSP) is designed to hold all platform-specific functionality, either because it relies on specific features of a target system, or because this provides the most efficient or flexible solution for the developer. For full details of its functions and macros, see the *HCC Base Platform Support Package User Guide*.

The RNDIS module makes use of the following standard PSP functions:

Function	Package	Component	Description
psp_memcpy()	psp_base	psp_string	Copies a block of memory. The result is a binary copy of the data.
psp_memset()	psp_base	psp_string	Sets the specified area of memory to the defined value.

The RNDIS module makes use of the following standard PSP macro:

Macro	Package	Component	Description
PSP_RD_LE32	psp_base	psp_endianness	Reads a 32 bit value that is to be interpreted as little-endian.