Point to Point Protocol User Guide

Version 1.10

For use with Point to Point Protocol (PPP) module versions 1.12 and above

Exported on 10/05/2018

Table of Contents

1 System Overview 5
1.1 Introduction 6
1.2 Feature Check 7
1.3 Packages and Documents 8
Packages 8
Documents 8
1.4 Change History 9
2 Source File List 10
2.1 API Header Files 10
2.2 Configuration Files 10
2.3 Source Code Files 11
2.4 Version File 11
2.5 Platform Support Package (PSP) Files 12
3 Configuration Options 13
3.1 config_ip_ppp.h 14
3.2 config_ip_ppp_chap.h 15
Static Options 15
Default Option 15
3.3 config_ip_ppp_ipcp.h 16
Static Options 16
Default Options 16
3.4 config_ip_ppp_lcp.h 19
Static Options 19
Default Options 20
3.5 config_ip_ppp_pap.h 21
Static Options 21
Default Options 21
3.6 config_ip_ppp_vjc.h 22
Static Option 22
Default Option 22
4 Serial Driver Configuration and Use 23
4.1 config_ip_ppp_driver_ser.h 24

4.2 config_ip_ppp_driver_ser.c 25
4.3 PPP Serial Driver API 25
4.4 Configuration Example 26
Overview and Assumptions 26
config_ip_ppp_driver_ser.h options 27
config_ip_ppp_driver_ser.c arrays 28
config_psp_ppp_uart.h options 28
config_psp_ppp_usbh_cdc_acm.h options 28
Initializing and Starting Operation 29
5 Application Programming Interface 31
5.1 Module Management 31
ip_ppp_init 32
ip_ppp_start 33
ip_ppp_stop 34
ip_ppp_delete 35
5.2 PPP and Driver Management 36
ip_ppp_connect 37
ip_ppp_disconnect 38
ip_ppp_drvser_init 39
ip_ppp_get_config 40
ip_ppp_set_config 41
ip_ppp_state 42
ip_ppp_register_ntf 43
5.3 Interface Management 44
ip_ppp_ifc_init 45
ip_ppp_ifc_start 46
ip_ppp_ifc_stop 47
ip_ppp_ifc_delete 48
5.4 Error Codes 49
5.5 Types and Definitions 50
PPP States 50
t_ip_ppp_config 51
t_ip_ppp_ntf_fn 52
6 Integration 53
6.1 OS Abstraction Layer 53
6.2 Utilities 53

6.3 PSP Porting 54
psp_ppp_ser_init 56
psp_ppp_ser_start 57
psp_ppp_ser_stop 58
psp_ppp_ser_delete 59
ppp_ser_tx_advance 60
psp_ppp_ser_rx 61
psp_ppp_ser_tx 62
psp_ppp_ser_tx_ch 63

1 System Overview

This chapter contains the fundamental information for this module.

The component sections are as follows:

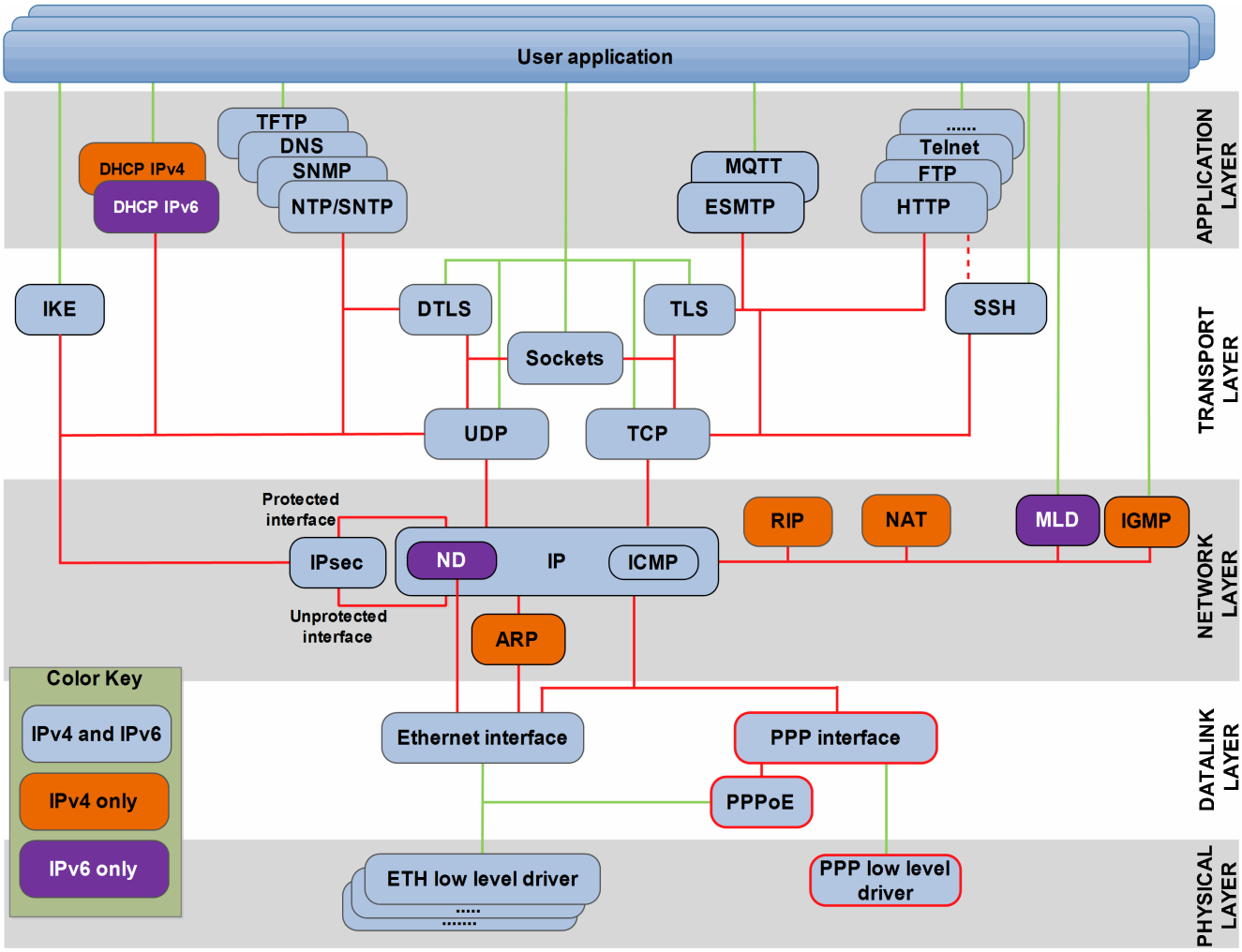
- Introduction – describes the main elements of the module.
- Feature Check – summarizes the main features of the module as bullet points.
- Packages and Documents – the *Packages* section lists the packages that you need in order to use this module. The *Documents* section lists the relevant user guides.
- Change History – lists the earlier versions of this manual, giving the software version that each manual describes.

Note: To download this manual as a PDF, see TCP/IP PDFs.

1.1 Introduction

This guide is for those who want to add a Point to Point Protocol (PPP) network interface to their HCC network stack. This lets you set up a network connection over a serial link to a remote network stack. An additional module (PPP over Ethernet or PPPoE) is required to run the PPP module on an Ethernet network.

The PPP module is part of the HCC MISRA-compliant TCP/IP stack, as shown below. (In this diagram green lines show interfaces available to users of the stack, red lines show interfaces internal to the TCP/IP system.)



This shows the three main PPP components:

- The PPP interface.
- The PPP over Ethernet (PPPoE) module.
- The PPP low level driver – this implements a serial driver for the HDLC framing protocol specified in RFC 1662 and used by the PPP implementation over RS232.

1.2 Feature Check

The main features of the system are the following:

- Conforms to the HCC Advanced Embedded Framework.
- Compliant with the HCC MISRA TCP/IP stack.
- Designed for integration with both RTOS and non-RTOS based systems.
- Compliant with RFC 1662.
- Support for up to four PPP interfaces.

The following options are configurable:

- PPP configuration options.
- Challenge Handshake Authentication Protocol (CHAP).
- IP Control Protocol (IPCP).
- Link Control Protocol (LCP).
- PPP Authentication Protocol (PAP).
- Van Jacobson (VJ) compression.
- Serial driver.

1.3 Packages and Documents

Packages

The table below 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.
ip_ppp	The PPP package described in this manual.
mip_base	The TCP/IP stack base package.
psp_template_uart	The Platform Support Package (PSP) that contains the Universal Asynchronous Receiver/Transmitter (UART) functions and macros used by the serial driver.

Documents

For an overview of HCC's TCP/IP stack software, 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 TCP/IP Dual Stack System User Guide

This is the core document that describes the complete TCP/IP stack. It covers both IPv4 and IPv6 systems.

HCC Point to Point Protocol User Guide

This is this document.

1.4 Change History

This section describes past changes to this manual.

- To download this manual or a PDF describing an earlier software version, see TCP/IP PDFs.
- For the history of changes made to the package code itself, see History: ip_ppp.

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	2018-10-05	1.14	Added *t_ip_ppp_ntf_fn* typedef. Added **ppp_ser_tx_advance()** to *PSP Porting*.
1.00	2018-06-27	1.12	First online version.

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 files and PSP files.

2.1 API Header Files

These files in the directory **src/api** are the only files that should be included by an application using this module. For details of these API functions, see Application Programming Interface.

File	Description
api_ip_ifc_ppp.h	Interface functions.
api_ip_ppp.h	PPP module functions.
api_ip_ppp_driver_ser.h	The single serial driver function.

2.2 Configuration Files

These files in the directory **src/config** contain all the configurable parameters of the system. Configure these as required. For details of these options, see Configuration Options.

File	Description
config_ip_ppp.h	System configuration options.
config_ip_ppp_chap.h	Challenge Handshake Authentication Protocol (CHAP) configuration options.
config_ip_ppp_driver_ser.c	Serial driver unit IDs.
config_ip_ppp_driver_ser.h	Serial driver configuration options.
config_ip_ppp_ipcp.h	IP Control Protocol (IPCP) configuration options.
config_ip_ppp_lcp.h	Link Control Protocol (LCP) configuration options.
config_ip_ppp_pap.h	PPP Authentication Protocol (PAP) configuration options.
config_ip_ppp_vjc.h	Van Jacobson compression options.

2.3 Source Code Files

These files in the directory **src/ip/stack/ppp** are the source code files. **These files should only be modified by HCC**.

File	Description
core/chap.c and **chap.h**	CHAP source code and header files.
core/ipcp.c and **ipcp.h**	IPCP source code and header files.
core/lcp.c and **lcp.h**	LCP source code and header files.
core/pap.c and **pap.h**	PAP source code and header files.
core/ppp.c and **ppp.h**	System configuration options.
core/vjc.c and **vjc.h**	Van Jacobson compression files.
driver/ppp_driver_ser.c, **driver/ppp_driver_ser_const.c** and **ppp_driver_ser.h**	Serial driver source code and header file.
interface/ifc_ppp.c	PPP interface source code.

2.4 Version File

The file **src/version/ver_ip_ppp.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.

2.5 Platform Support Package (PSP) Files

These files in the directory **src/psp/target/ppp_driver_ser** provide functions and elements the core code needs to use, depending on the hardware. Modify these files as required for your hardware.

Note:

- These are PSP implementations for the specific microcontroller and board; you may need to modify these to work with a different microcontroller and/or development board. See PSP Porting for details.
- In the package these files are offset to avoid overwriting an existing implementation. Copy them to the root **hcc** directory for use.

File	Description
psp_ppp_driver_ser.c and **.h**	Serial driver code and header file.
usbh_cdc_acm/ psp_ppp_driver_ser_usbh_cdc_acm.c and **.h**	USB host CDC-ACM serial PPP driver code and header file.
config/config_psp_ppp_usbh_cdc_acm.h	CDC-ACM configuration options.
uart/psp_ppp_driver_ser_uart.c and **.h**	UART Serial PPP driver code and header file.
config/config_psp_ppp_uart.h	UART configuration options.

3 Configuration Options

Set the system configuration options in the files described in this section. The available configuration options and their default values are described.

The following options are configurable:

- PPP configuration options
- Challenge Handshake Authentication Protocol (CHAP)
- IP Control Protocol (IPCP)
- Link Control Protocol (LCP)
- PPP Authentication Protocol (PAP)
- Van Jacobson (VJ) compression
- Serial driver - for this, see Serial Driver Configuration and Use.

3.1 config_ip_ppp.h

Set the system configuration options in the file **src/config/config_ip_ppp.h**. You can change any option for an interface at run time by using **ip_ppp_set_config()** to modify the *t_ip_ppp_config* structure.

IP_PPP_DEFAULT_ROLE

The default role in the PPP connection. The default of 0 means client. Set this to 1 for a server.

This is *b_ipppc_server* in the *t_ip_ppp_config* structure.

IP_PPP_DEFAULT_WIN_HS

Set this to 1 if Windows handshake is to be performed. The default of 0 means Windows handshake is not performed.

This is *b_ipppc_win_hs* in the *t_ip_ppp_config* structure.

IP_PPP_CHAP_ENABLE

Set this to 1 if CHAP is supported. This enables CHAP on all PPP interfaces but CHAP can be enabled/disabled for any interface by using *b_ipppc_chap* in the *t_ip_ppp_config* structure.

The default value is 0, disabling support for CHAP; the CHAP module is not compiled in this case and *b_ipppc_chap* is not available.

IP_PPP_VJC_ENABLE

Keep the default of 1 if Van Jacobson compression of TCP/IP headers is supported. This enables VJ compression on all PPP interfaces but it can be enabled/disabled for any interface by using *b_ipppc_vjc* in the *t_ip_ppp_config* structure.

To disable support for VJ compression, set this to 0; the VJ compression module is not compiled in this case and *b_ipppc_vjc* is not available.

3.2 config_ip_ppp_chap.h

Set the Challenge Handshake Authentication Protocol configuration options in the file **src/config/config_ip_ppp_chap.h**.

Static Options

The following options are fixed for all PPP instances.

CHAP_SECRET_MAX_LEN

The length of the CHAP secret. This should be no longer than 63. The default value is 8.

CHAP_EXPIRY_TIME

The value of the CHAP expiry counter in seconds. The default value is 3.

CHAP_MAX_RETRY

The number of authenticate request retries. The default value is 2.

CHAP_MD5_INPUT_LEN

The length of the buffer containing the message hashed by the MD5 algorithm. The default value is 128.

CHAP_CHALLENGE_LEN

The length of the CHAP challenge sent to the peer. This must be no longer than 64. The default value is 16.

Default Option

You can change the following options for an interface at run time by using **ip_ppp_set_config()** to modify the *t_ip_ppp_config* structure.

CHAP_DEFAULT_SECRET

The default CHAP secret. This should be no longer than 63. The default value is "brown".

This is *p_ipppc_chap_secret* in the *t_ip_ppp_config* structure.

3.3 config_ip_ppp_ipcp.h

Set the IP Control Protocol configuration options in the file **src/config/config_ip_ppp_ipcp.h**.

Static Options

The following are fixed for all PPP instances.

IPCP_EXPIRY_TIME

The value of the IPCP expiry counter in seconds. The default value is 3.

IPCP_MAX_CONFIG_RETRY

The number of configuration request retries. The default value is 2.

IPCP_MAX_REJECT_OPTS

The size of the buffer storing rejected options. The default value is 32.

Default Options

You can change the following options for an interface at run time by using **ip_ppp_set_config()** to modify the *t_ip_ppp_config* structure.

> **Note:** The following server options are repeated for each PPP unit supported (up to 4). The "_0" is replaced by "_1" for the second set, "_2" for the third set, and so on.

IPCP_DEFAULT_MASK_0

The default network mask. The default value is `{ 0xFFu, 0xFFu, 0xFFu, 0x00u }`

This is *p_ipppc_ipcp_default_mask* in the *t_ip_ppp_config* structure.

IPCP_LOCAL_ADDR_0

The local IP address. The default value is `{ 0xC0, 0xA8, 0x0F, 0x8A }`

This is *p_ipppc_ipcp_local_addr* in the *t_ip_ppp_config* structure.

IPCP_REMOTE_ADDR_0

The remote IP address. The default value is `{ 0xC0, 0xA8, 0x0F, 0x01 }`

This is *p_ipppc_ipcp_local_addr* in the *t_ip_ppp_config* structure.

IPCP_DNS1_ADDR_0

The DNS1 address. The default value is `{ 0x00, 0x00, 0x00, 0x00 }`

This is *p_ipppc_ipcp_dns1_addr* in the *t_ip_ppp_config* structure.

IPCP_DNS2_ADDR_0

The DNS2 address. The default value is `{ 0x00, 0x00, 0x00, 0x00 }`

This is *p_ipppc_ipcp_dns2_addr* in the *t_ip_ppp_config* structure.

IPCP_DEFAULT_MASK_1

The default network mask. The default value is `{ 0xFFu, 0xFFu, 0xFFu, 0x00u }`

IPCP_LOCAL_ADDR_1

The local IP address. The default value is `{ 0xC0, 0xA8, 0x0E, 0x8B }`

IPCP_REMOTE_ADDR_1

The remote IP address. The default value is `{ 0xC0, 0xA8, 0x0E, 0x02 }`

IPCP_DNS1_ADDR_1

The DNS1 address. The default value is `{ 0x00, 0x00, 0x00, 0x00 }`

IPCP_DNS2_ADDR_1

The DNS2 address. The default value is `{ 0x00, 0x00, 0x00, 0x00 }`

IPCP_DEFAULT_MASK_2

The default network mask. The default value is `{ 0xFFu, 0xFFu, 0xFFu, 0x00u }`

IPCP_LOCAL_ADDR_2

The local IP address. The default value is `{ 0xC0, 0xA8, 0x0D, 0x8C }`

IPCP_REMOTE_ADDR_2

The remote IP address. The default value is `{ 0xC0, 0xA8, 0x0D, 0x03 }`

IPCP_DNS1_ADDR_2

The DNS1 address. The default value is `{ 0x00, 0x00, 0x00, 0x00 }`

IPCP_DNS2_ADDR_2

The DNS2 address. The default value is `{ 0x00, 0x00, 0x00, 0x00 }`

IPCP_DEFAULT_MASK_3

The default network mask. The default value is `{ 0xFFu, 0xFFu, 0xFFu, 0x00u }`

IPCP_LOCAL_ADDR_3

The local IP address. The default value is `{ 0xC0, 0xA8, 0x0C, 0x8D }`

IPCP_REMOTE_ADDR_3

The remote IP address. The default value is `{ 0xC0, 0xA8, 0x0C, 0x04 }`

IPCP_DNS1_ADDR_3

The DNS1 address. The default value is `{ 0x00, 0x00, 0x00, 0x00 }`

IPCP_DNS2_ADDR_3

The DNS2 address. The default value is `{ 0x00, 0x00, 0x00, 0x00 }`

3.4 config_ip_ppp_lcp.h

Set the Link Control Protocol configuration options in the file **src/config/config_ip_ppp_lcp.h**.

Static Options

The following options are fixed for all PPP instances.

LCP_MAX_RECEIVE_UNIT

The maximum size of received frames in bytes. The default value is 512.

LCP_ADDR_CTRL_COMPR

Keep this at the default of 1 to allow address and control field compression in received/sent HDLC frames.

LCP_PROTO_COMPR

Keep this at the default of 1 to allow protocol field compression in received/sent HDLC frames.

LCP_RX_ACCM

The value of the async control character map for the RX link. Use 0xFFFFFFFF if this is unknown. The default value is 0.

LCP_EXPIRY_TIME

The value of the LCP expiry counter in seconds. The default value is 3.

LCP_MAX_CONFIG_RETRY

The number of config request retries. The default value is 20.

LCP_MAX_TERMINATE_RETRY

The number of terminate request retries. The default value is 2.

LCP_MAX_REJECT_OPTS

The rejected options buffer size. The default value is 32.

Default Options

You can change the following options for an interface at run time by using **ip_ppp_set_config()** to modify the *t_ip_ppp_config* structure.

LCP_ADDR_CTRL_COMPR_ENABLE

Keep the default of 1 to allow address and control field compression in received/sent HDLC frames.

This is *b_ipppc_lcp_addr_comp* in the *t_ip_ppp_config* structure.

LCP_PROTO_COMPR_ENABLE

Keep the default of 1 to allow protocol field compression in received/sent HDLC frames.

This is *b_ipppc_lcp_proto_comp* in the *t_ip_ppp_config* structure.

LCP_DEFAULT_RX_ACCM

The default value of the asynchronous control character map for RX link. Use 0xFFFFFFFF if this is unknown. The default is 0.

This is *ipppc_lcp_rx_accm* in the *t_ip_ppp_config* structure.

3.5 config_ip_ppp_pap.h

Set the PPP Authentication Protocol configuration options in the file **src/config/config_ip_ppp_pap.h**.

Static Options

The following options are fixed for all PPP instances.

PAP_EXPIRY_TIME

The value of the PAP expiry counter in seconds. The default value is 3.

PAP_MAX_RETRY

The number of authenticate request retries. The default value is 2.

PAP_MAX_STRING_LEN

The maximum length of the ID or password. The default value is 16.

Default Options

You can change the following options for an interface at run time by using **ip_ppp_set_config()** to modify the *t_ip_ppp_config* structure.

PAP_DEFAULT_HOST_ID

The default PAP host id for all PPP interfaces. The default value is "".

This is *p_ipppc_pap_host_id* in the *t_ip_ppp_config* structure.

PAP_DEFAULT_HOST_PASSWORD

The default PAP host password for all PPP interfaces. The default value is "".

This is *p_ipppc_pap_host_password* in the *t_ip_ppp_config* structure.

PAP_DEFAULT_PEER_ID

The default PAP peer id for all PPP interfaces. The default value is "".

This is *p_ipppc_pap_peer_id* in the *t_ip_ppp_config* structure.

PAP_DEFAULT_PEER_PASSWORD

The default PAP peer password for all PPP interfaces. The default value is "".

This is *p_ipppc_pap_peer_password* in the *t_ip_ppp_config* structure.

3.6 config_ip_ppp_vjc.h

Set the Van Jacobson compression options in the file **src/config/config_ip_ppp_vjc.h**.

Static Option

The following option is fixed for all PPP instances.

VJC_NUM_CONN

The number of connection slots per PPP unit. The default value is 4.

Default Option

You can change the following options for an interface at run time by using **ip_ppp_set_config()** to modify the *t_ip_ppp_config* structure.

VJC_ALLOW_SLOT_ID_COMPR

Set this to 1 to enable connection ID compression. The default value is 0.

4 Serial Driver Configuration and Use

The PPP module includes a serial interface that can be used with the following types of serial communication media:

- UART
- HCC USB Host stack with CDC-ACM class driver
- Any other custom serial communication medium

To configure it properly, follow the guidelines in this section.

For higher level configuration, set the options/arrays in the following files:

- config_ip_ppp_driver_ser.h
- config_ip_ppp_driver_ser.c

For lower level configuration, the two PSP configuration files shown below are designed to be used as they are. You can, however, modify them if needed, for example to allocate the buffer pools in a specific way. One use case for this is the case when UART DMA is used and the linker has to be forced to place the UART buffer pools in an area that DMA transfers can be initiated to/from.

Note: As these files are part of the PSP they are allowed to contain platform-specific code (for example, pragmas). These are similar to the corresponding PSP modules (**psp_ppp_ser_uart** and **psp_ppp_ser_usbh_cdc_acm**).

File and option	Description
src\config\config_psp_ppp_uart.h PSP_PPP_UART_BUF_SIZE_n	The size of the buffer the IP stack uses for transfers on the PPP-UART interface Nr.n. Note that all MTU size buffers and minimal size buffers (see **config_ip.h**) are drawn from this pool.
src\config\config_psp_ppp_usbh_cdc_acm.h PSP_PPP_USBH_CDC_ACM_BUF_SIZE_x	The size of the buffer the IP stack uses for transfers on PPP-USBH-CDC-ACM interface Nr.x. Note that all MTU size buffers and minimal size buffers (see **config_ip.h**) are drawn from this pool.

4.1 config_ip_ppp_driver_ser.h

Set the serial interface configuration options in the file **src/config/config_ip_ppp_driver_ser.h**.

PPP_DRVSER_UART_UNIT_CNT

The number of PPP UART interfaces. The default value is 1.

PPP_DRVSER_USBH_CDC_ACM_UNIT_CNT

The number of PPP USBH_CDC_ACM interfaces. The default value is 0.

PPP_DRVSER_CUSTOM_UNIT_CNT

The number of PPP CUSTOM interfaces. The default value is 0.

> **Note:** The above three options give the value of PPP_UNIT_CNT shown below, the total number of PPP units supported. **Do not change this value.**

PPP_UNIT_CNT

The total number of PPP units supported:

```
 ( PPP_DRVSER_UART_UNIT_CNT\
+ PPP_DRVSER_USBH_CDC_ACM_UNIT_CNT\
 + PPP_DRVSER_CUSTOM_UNIT_CNT )
```

PPP_DRVSER_TASK_STACK_SIZE

The stack size of the PPP serial driver's task responsible for handling communication on all serial PPP instances. The default value is 512.

> **Note:** The following three parameters are repeated for each PPP unit supported (up to 4). The "_0" is replaced by "_1" for the second set and so on.

PPP_DRVSER_BAUD_RATE_0

The baud rate to use on serial PPP instance 0. The default value is 115200.

PPP_DRVSER_MTU_SIZE_0

The MTU size to use on serial PPP instance 0. The default value is 256.

PPP_DRVSER_RX_BUF_DESC_0

The number of Receive (RX) buffer descriptors the serial PPP instance can handle. The IP stack always adds this many RX buffers to this instance.. The default value is 16.

4.2 config_ip_ppp_driver_ser.c

This file defines unit IDs for UART, CDC-ACM, and custom units. It defines three arrays:

Array	Description
g_ppp_ser_uart_unit [PPP_DRVSER_UART_UNIT_CNT]	Assigns UART unit IDs to PPP-UART units.
g_ppp_ser_usbh_cdc_acm_unit [PPP_DRVSER_USBH_CDC_ACM_UNIT_CNT]	Assigns USBH-CDC-ACM unit IDs to PPP-USBH-CDC-ACM units.
g_ppp_ser_custom_unit [PPP_DRVSER_CUSTOM_UNIT_CNT]	Assigns custom unit IDs to PPP-Custom units.

4.3 PPP Serial Driver API

The file **src/api/api_ip_ppp_driver_ser.h** defines the following macros that calculate the parameter *param* that is passed to ip_ppp_drvser_init() when **ip_ppp_ifc_init()** is called to initialize a PPP interface.

Macro	Description
IP_PPP_DRVSER_PARAM_UART (ppp_uart_id)	The macro to use when calling **ip_ppp_ifc_init()** to identify the PPP serial unit based on the PPP-UART unit ID.
IP_PPP_DRVSER_PARAM_USBH_CDC_ACM (ppp_usbh_cdc_acm_id)	The macro to use when calling **ip_ppp_ifc_init()** to identify the PPP serial unit based on the PPP-USBH-CDC_ACM unit ID.
IP_PPP_DRVSER_PARAM_CUSTOM (ppp_custom_id)	The macro to use when calling **ip_ppp_ifc_init()** to identify the PPP serial unit based on the PPP-Custom unit ID.

4.4 Configuration Example

Overview and Assumptions

This example assumes that the system has the following components:

Three UART ports, handled by the same UART driver (**hcc\src\psp\target\uart\psp_uart.c**):

- UART0
- UART1
- UART2

USBH-CDC-ACM configured to handle a maximum of four units (four CDC-ACM data channels. Note that one device can have more channels. That is, it is also possible to handle more CDC-ACM USB devices connected to more host ports, or to the ports of an external hub connected to the host port.)

- CDC-ACM0
- CDC-ACM1
- CDC-ACM2
- CDC-ACM3

Three PPP network interfaces:

- IFC0: communicating with a modem on UART1.
- IFC1: communicating with a Linux PC on UART2.
- IFC2: communicating with a PPP device on USBH-CDC-ACM3.

In this example UART0 is used, for example for a command line interface, and CDC-ACM0 to CDC-ACM2 channels of the device expected on the USBH port are not receiving PPP connections but are acting as raw serial communication channels.

config_ip_ppp_driver_ser.h options

The file **src\config\config_ip_ppp_driver_ser.h** is set up as follows:

Option	Value	Description
PPP_DRVSER_UART_UNIT_CNT	2	The number of PPP UART interfaces.
PPP_DRVSER_USBH_CDC_ACM_UNIT_CNT	1	The number of PPP USBH_CDC_ACM interfaces.
PPP_DRVSER_CUSTOM_UNIT_CNT	0	The number of PPP CUSTOM interfaces.
PPP_DRVSER_TASK_STACK_SIZE	512	The default value is probably adequate; this might depend on the RTOS used.
PPP_DRVSER_BAUD_RATE_0	115200	The baud rate of the modem.
PPP_DRVSER_MTU_SIZE_0	256	An MTU size supported by the modem.
PPP_DRVSER_RX_BUF_DESC_0	16	An optimal number of Rx descriptors, considering that all all transfers (Rx, Tx, MTU size, and minimum size) will be using the PPP instance's buffer pool.
PPP_DRVSER_BAUD_RATE_1	115200	The baud rate of the communication. Note that **the same baud rate must be used on the Linux machine**.
PPP_DRVSER_MTU_SIZE_1	256	An MTU size supported by the Linux side.
PPP_DRVSER_RX_BUF_DESC_1	16	An optimal number of Rx descriptors, considering that all all transfers (Rx, Tx, MTU size, minimum size) will be using the PPP instance's buffer pool.
PPP_DRVSER_BAUD_RATE_2	115200	The Baud rate of the communication - note that in case the CDC-ACM device used is a USB-to-RS232 type device then this will be the baud rate used on the RS232 side. If the device does not convert USB CDC-ACM data to RS232 but processes it itself, then this baud rate will not have any effect.
PPP_DRVSER_MTU_SIZE_2	256	An MTU size supported by the other side.
PPP_DRVSER_RX_BUF_DESC_2	16	An optimal number of Rx descriptors, considering that all all transfers (Rx, Tx, MTU size, and minimum size) will use the PPP instance's buffer pool.

config_ip_ppp_driver_ser.c arrays

The file **src\config\config_ip_ppp_driver_ser.c** is set up as follows:

```
g_ppp_ser_uart_unit[PPP_DRVSER_UART_UNIT_CNT] =
{
    1u
    , 2u
};

g_ppp_ser_usbh_cdc_acm_unit[PPP_DRVSER_USBH_CDC_ACM_UNIT_CNT] =
{
    3u
};
```

config_psp_ppp_uart.h options

The file **src\config\config_psp_ppp_uart.h** is set up as follows:

Option	Value	Description
PSP_PPP_UART_BUF_SIZE_0	8192	The size of the buffer the IP stack uses for transfers on the PPP-UART interface 0.
PSP_PPP_UART_BUF_SIZE_1	8192	The size of the buffer the IP stack uses for transfers on the PPP-UART interface 1.

config_psp_ppp_usbh_cdc_acm.h options

The file **src\config\config_psp_ppp_usbh_cdc_acm.h** is set up as follows:

Option	Value	Description
PSP_PPP_USBH_CDC_ACM_BUF_SIZE_0	8192	The size of the buffer the IP stack uses for transfers on PPP-USBH-CDC-ACM interface 0.

Initializing and Starting Operation

Do the following:

1. Initialize the IP stack - see the IP stack documentation.

2. Initialize the PPP component:

```
ip_ppp_init();
```

3. Initialize the PPP network interfaces, in three steps:

3a. Initialize the PPP interface using the PPP serial driver with PPP-UART instance 0 (as the default gateway interface):

```
 ip_ppp_ifc_init( ip_ppp_drvser_init
                  , IP_PPP_DRVSER_PARAM_UART( 0u )
                  , IP_IFC_OPT_DEF_GATEWAY
                  , &g_ip_ppp_uart0_ifc_hdl );
```

3b. Initialize the PPP interface using the PPP serial driver with PPP-UART instance 1:

```
ip_ppp_ifc_init( ip_ppp_drvser_init
                 , IP_PPP_DRVSER_PARAM_UART( 1u )
                 , 0u
                 , &g_ip_ppp_uart1_ifc_hdl );
```

3c. Initialize the PPP interface using the PPP serial driver with PPP-USBH-CDCACM instance 0:

```
ip_ppp_ifc_init( ip_ppp_drvser_init
                 , IP_PPP_DRVSER_PARAM_USBH_CDC_ACM( 0u )
                 , 0u
                 , &g_ip_ppp_cdcacm0_ifc_hdl );
```

4. Configure the PPP interfaces if necessary:

```
t_ip_config  ip_config_ppp;

ip_config.ipc_ipaddr = ...
        ...
ip_set_config( g_ip_ppp_uart0_ifc_hdl, &ip_config, IP_CONFIGURE_IP_V4 );
        ...
ip_config.ipc_ipaddr = ...
        ...
ip_set_config( g_ip_ppp_uart1_ifc_hdl, &ip_config, IP_CONFIGURE_IP_V4 );

ip_config.ipc_ipaddr = ...
        ...
ip_set_config( g_ip_ppp_cdcacm0_ifc_hdl, &ip_config, IP_CONFIGURE_IP_V4 );
```

5. Start the PPP module:

```
ip_ppp_start();
```

6. Start the PPP network interfaces:

```
ip_ppp_ifc_start( g_ip_ppp_uart0_ifc_hdl );

ip_ppp_ifc_start( g_ip_ppp_uart1_ifc_hdl );

ip_ppp_ifc_start( g_ip_ppp_cdcacm0_ifc_hdl );
```

7. Connect to the PPP peer if necessary:

```
ip_ppp_connect( g_ip_ppp_uart0_ifc_hdl );

ip_ppp_connect( g_ip_ppp_uart1_ifc_hdl );

ip_ppp_connect( g_ip_ppp_cdcacm0_ifc_hdl );
```

5 Application Programming Interface

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

5.1 Module Management

The functions are the following:

Function	Description
ip_ppp_init()	Initializes the module and allocates the required resources.
ip_ppp_start()	Starts the module.
ip_ppp_stop()	Stops the module.
ip_ppp_delete()	Deletes the module and releases the resources it used.

ip_ppp_init

Use this function to initialize the module and allocate the required resources.

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

Format

```
t_ip_ret ip_ppp_init ( void )
```

Arguments

Argument
None.

Return values

Return value	Description
IP_SUCCESS	Successful execution.
IP_ERROR	Operation failed.

ip_ppp_start

Use this function to start the PPP module.

> **Note:** You must call **ip_ppp_init()** before this.

Format

```
t_ip_ret ip_ppp_start ( void )
```

Arguments

Argument
None.

Return values

Return value	Description
IP_SUCCESS	Successful execution.
IP_ERROR	Operation failed.

ip_ppp_stop

Use this function to stop the PPP module.

Format

```
t_ip_ret ip_ppp_stop ( void )
```

Arguments

Argument
None.

Return values

Return value	Description
IP_SUCCESS	Successful execution.
IP_ERROR	Operation failed.

ip_ppp_delete

Use this function to delete the PPP module and release the associated resources.

Format

```
t_ip_ret ip_ppp_delete ( void )
```

Arguments

Argument
None.

Return values

Return value	Description
IP_SUCCESS	Successful execution.
IP_ERROR	Operation failed.

5.2 PPP and Driver Management

The functions are the following:

Function	Description
ip_ppp_connect()	Initiates the PPP connection sequence on the given interface.
ip_ppp_disconnect()	Disconnects PPP on the given interface.
ip_ppp_drvser_init()	Initializes the serial driver.
ip_ppp_get_config()	Gets an interface's PPP configuration.
ip_ppp_set_config()	Configures PPP on an interface.
ip_ppp_state()	Gets the current PPP connection state.
ip_ppp_register_ntf()	Registers a notification function to be called by the PPP stack to get the connection state.

ip_ppp_connect

Use this function to initiate the PPP connection sequence on the given interface.

Format

```
t_ip_ret ip_ppp_connect ( const t_ip_ifc_hdl ifc_hdl )
```

Arguments

Argument	Description	Type
ifc_hdl	The interface handle.	t_ip_ifc_hdl

Return values

Return value	Description
IP_SUCCESS	Successful execution.
IP_ERR_INVALID_STATE	The interface is already connected.

ip_ppp_disconnect

Use this function to disconnect PPP on the given interface.

Format

```
t_ip_ret ip_ppp_disconnect ( const t_ip_ifc_hdl ifc_hdl )
```

Arguments

Argument	Description	Type
ifc_hdl	The interface handle.	t_ip_ifc_hdl

Return values

Return value	Description
IP_SUCCESS	Successful execution.
IP_ERR_INVALID_STATE	The state is not connected.

ip_ppp_drvser_init

Use this function to initialize the serial driver. Pass this function to **ip_ppp_ifc_init()** when initializing a PPP interface.

Format

```
t_nwdriver_ret ip_ppp_drvser_init (
    uint32_t               param,
    t_nwdriver * * const   p_driver )
```

Arguments

Argument	Description	Type
param	The parameter.	uint32_t
p_driver	A pointer to the serial driver structure.	t_nwdriver * *

Return values

Return value	Description
NWDRIVER_SUCCESS	Successful execution.
NWDRIVER_ERROR	Operation failed .

ip_ppp_get_config

Use this function to get an interface's PPP configuration.

Format

```
t_ip_ret ip_ppp_get_config (
    const t_ip_ifc_hdl    ifc_hdl,
    t_ip_ppp_config *     p_ip_ppp_config )
```

Arguments

Argument	Description	Type
ifc_hdl	The interface handle.	t_ip_ifc_hdl
p_ip_ppp_config	Where to write the configuration.	t_ip_ppp_config *

Return values

Return value	Description
IP_ERR_INVALID_STATE	State is not disconnected.
ELSE	Values returned by **ip_ifc_check()**.

ip_ppp_set_config

Use this function to configure PPP on an interface.

Format

```
t_ip_ret ip_ppp_set_config (
    const t_ip_ifc_hdl   ifc_hdl,
    t_ip_ppp_config *    p_ip_ppp_config )
```

Arguments

Argument	Description	Type
ifc_hdl	The interface handle.	t_ip_ifc_hdl
p_ip_ppp_config	A pointer to the configuration to set.	t_ip_ppp_config *

Return values

Return value	Description
IP_ERR_INVALID_STATE	State is not disconnected.
ELSE	Values returned by **ip_ifc_check()**.

ip_ppp_state

Use this function to get the current PPP connection state.

Format

```
t_ip_ret ip_ppp_state (
    const t_ip_ifc_hdl   ifc_hdl,
    uint8_t * const      p_state )
```

Arguments

Argument	Description	Type
ifc_hdl	The interface handle.	t_ip_ifc_hdl
p_state	Where to write the state.	uint8_t *

Return values

Return value	Description
IP_SUCCESS	Successful execution.
IP_ERROR	Operation failed.

ip_ppp_register_ntf

Use this function to register a notification function to be called by the PPP stack to get the connection state.

Note: It is the user's responsibility to provide any notification functions required by the application. Providing such functions is optional.

Format

```
t_ip_ret ip_ppp_register_ntf (
    const t_ip_ifc_hdl   ifc_hdl,
    t_ip_ppp_ntf_fn      ip_ppp_ntf_fn )
```

Arguments

Parameter	Description	Type
ifc_hdl	The interface handle.	t_ip_ifc_hdl
ip_ppp_ntf_fn	The notification function to use when an event occurs.	t_ip_ppp_ntf_fn

Return Values

Return value	Description
IP_SUCCESS	Successful execution.

5.3 Interface Management

The functions are the following:

Function	Description
ip_ppp_ifc_init()	Initializes the PPP interface and allocates the required resources.
ip_ppp_ifc_start()	Starts the PPP interface.
ip_ppp_ifc_stop()	Stops the PPP interface.
ip_ppp_ifc_delete()	Deletes the PPP interface and releases the resources it used.

ip_ppp_ifc_init

Use this function to initialize the PPP interface and allocate the required resources.

Note: You must call this function before any other interface function.

Format

```
t_ip_ret ip_ppp_ifc_init (
    const t_nwdriver_init   p_lb_drv_init,
    const uint32_t          param,
    const uint8_t           options,
    t_ip_ifc_hdl * const    p_ifc_hdl )
```

Arguments

Argument	Description	Type
p_lb_drv_init	A pointer to the PPP driver init function.	t_nwdriver_init
param	The parameter to pass to the driver init function.	uint32_t
options	One of the IP configuration options, either: • Enable DHCP for this interface. • Set this interface to be the default gateway.	uint8_t
p_ifc_hdl	Where to write the interface handle.	t_ip_ifc_hdl *

Return values

Return value	Description
IP_SUCCESS	Successful execution.
IP_ERROR	Operation failed.

ip_ppp_ifc_start

Use this function to start the PPP interface.

> **Note:** You must call **ip_ppp_ifc_init()** before this to initialize the interface.

Format

```
t_ip_ret ip_ppp_ifc_start ( const t_ip_ifc_hdl ifc_hdl )
```

Arguments

Argument	Description	Type
ifc_hdl	The interface handle.	t_ip_ifc_hdl

Return values

Return value	Description
IP_SUCCESS	Successful execution.
IP_ERROR	Operation failed.

ip_ppp_ifc_stop

Use this function to stop the PPP interface.

Format

```
t_ip_ret ip_ppp_ifc_stop ( const t_ip_ifc_hdl ifc_hdl )
```

Arguments

Argument	Description	Type
ifc_hdl	The interface handle.	t_ip_ifc_hdl

Return values

Return value	Description
IP_SUCCESS	Successful execution.
IP_ERROR	Operation failed.

ip_ppp_ifc_delete

Use this function to delete the interface and release the associated resources.

Format

```
t_ip_ret ip_ppp_ifc_delete ( const t_ip_ifc_hdl ifc_hdl )
```

Arguments

Argument	Description	Type
ifc_hdl	The interface handle.	t_ip_ifc_hdl

Return values

Return value	Description
IP_SUCCESS	Successful execution.
IP_ERROR	Operation failed.

5.4 Error Codes

The table below lists the return codes that may be generated by the API calls.

Error code	Value	Meaning
IP_SUCCESS	0	Successful execution.
IP_ERROR	4	Operation failed.
IP_ERR_INVALID_STATE	20	The state is not valid for the connect/disconnect function. For example, it was already connected when you called **ip_ppp_connect()**.

5.5 Types and Definitions

The PPP states are defined in the API Header file **api_ip_ifc_ppp.h**.

PPP States

The possible states are as as follows.

Name	Value	Description
IP_PPP_STATE_CONNECTED	1	PPP is connected.
IP_PPP_STATE_DISCONNECTED	2	PPP is disconnected.
IP_PPP_STATE_CONNECTING	3	PPP is connecting.
IP_PPP_STATE_DISCONNECTING	4	PPP is disconnecting.

t_ip_ppp_config

The *t_ip_ppp_config* structure holds an interface configuration. It has the following elements:

Element	Type	Description
Global parameters		
b_ipppc_server	uint8_t	The role: 1 for server, 0 for client.
b_ipppc_win_hs	uint8_t	1 to perform Windows handshake.
CHAP parameters (if IP_PPP_CHAP_ENABLE is set)		
b_ipppc_chap	uint8_t	CHAP enabled.
p_ipppc_chap_secret	char_t *	The CHAP secret.
VJC parameter (if IP_PPP_VJC_ENABLE is set)		
b_ipppc_vjc	uint8_t	VJC enabled.
LCP parameters		
ipppc_lcp_rx_accm	uint32_t	The asynchronous control character map for RX link.
b_ipppc_lcp_addr_comp	uint8_t	TRUE if protocol field compression is enabled, otherwise FALSE.
b_ipppc_lcp_proto_comp	uint8_t	TRUE if address field compression is enabled, otherwise FALSE.
PAP parameters		
p_ipppc_pap_host_id	char_t *	The PAP host ID.
p_ipppc_pap_host_password	char_t *	The PAP host password.
p_ipppc_pap_peer_id	char_t *	The PAP peer ID.
p_ipppc_pap_peer_password	char_t *	The PAP peer password.
IPCP parameters		
p_ipppc_ipcp_default_mask	uint8_t *	The IPCP default mask.
p_ipppc_ipcp_local_addr	uint8_t *	The IPCP local address.
p_ipppc_ipcp_remote_addr	uint8_t *	The IPCP remote address.
p_ipppc_ipcp_dns1_addr	uint8_t *	The DNS1 address.

Element	Type	Description
p_ipppc_ipcp_dns2_addr	uint8_t *	The DNS2 address.

t_ip_ppp_ntf_fn

The *t_ip_ppp_ntf_fn* typedef defines the notification function that reports a state.

Format

```
typedef void ( * t_ip_ppp_ntf_fn ) (
    const t_ip_ifc_hdl   ifc_hdl,
    const uint8_t        state )
```

Arguments

Element	Type	Description
ifc_hdl	t_ip_ifc_hdl	The interface handle.
state	uint8_t	The state.

6 Integration

This section describes all aspects of the DHCP module that require integration with your target project. This includes porting and configuration of external resources.

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.

This module uses the following OAL components:

OAL Resource	Number Required
Tasks	1
Mutexes	1
Events	1

6.2 Utilities

The PPP code creates and uses a single timer in the **hcc_timer** module.

The **hcc_timer** module is included in your system when you install the base TCP/IP modules.

6.3 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 *Base Platform Support Package User Guide* .

The module makes use of the following standard PSP functions:

Function	Package	Element	Description
psp_getrand()	psp_base	psp_rand	Generates a random number.
psp_memcmp()	psp_base	psp_string	Compares two blocks of memory.
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.
psp_strncat()	psp_base	psp_string	Appends a string.
psp_strncpy()	psp_base	psp_string	Copies one string of defined length to another.
psp_strncmp()	psp_base	psp_string	Compares two strings of defined length.
psp_strnlen()	psp_base	psp_string	Gets the length of a string.

The module makes use of the following standard PSP UART functions. For details of these functions, see the UART Driver PSP User Guide.

Function	Package	Element	Description
psp_uart_init()	psp_base	psp_uart	Initializes a UART port.
psp_uart_start()	psp_base	psp_uart	Starts a UART port.
psp_uart_stop()	psp_base	psp_uart	Stops a UART port.
psp_uart_delete()	psp_base	psp_uart	Deletes a UART port.
psp_uart_rx()	psp_base	psp_uart	Receives data over the UART.
psp_uart_tx()	psp_base	psp_uart	Sends data over the UART.
psp_uart_tx_ch()	psp_base	psp_uart	Writes one character over the UART.

The module also uses PSP_UART_xxx macros like PSP_UART_BITS_8 and PSP_UART_FLAG_RX. For details of these, see the UART Driver PSP User Guide.

The module makes use of the following standard PSP macros:

Macro	Package	Element	Description
PSP_RD_BE16	psp_base	psp_endianness	Reads a 16 bit value stored as big-endian from a memory location.
PSP_RD_BE32	psp_base	psp_endianness	Reads a 32 bit value stored as big-endian from a memory location.
PSP_RD_LE32	psp_base	psp_endianness	Reads a 32 bit value stored as little-endian from a memory location.
PSP_WR_BE16	psp_base	psp_endianness	Writes a 16 bit value to be stored as big-endian to a memory location.
PSP_WR_BE32	psp_base	psp_endianness	Writes a 32 bit value to be stored as big-endian to a memory location.
PSP_WR_LE32	psp_base	psp_endianness	Writes a 32 bit value to be stored as little-endian to a memory location.

The following functions are provided by the PSP to perform various tasks. They are designed for a specific microcontroller and development board. You may need to port them to work with your hardware solution; they are designed to make porting easy.

The package includes samples in the **psp_ppp_driver_ser_uart.c** and **psp_ppp_driver_ser_usbh_cdc_acm.c** files.

Function	Description
psp_ppp_ser_init()	Initializes a PPP port.
psp_ppp_ser_start()	Starts a PPP port.
psp_ppp_ser_stop()	Stops a PPP port.
psp_ppp_ser_delete()	Deletes a PPP port.
ppp_ser_tx_advance()	Advance TX transfer. This function is called from the ppp_serial driver's transfer task when dealing with completed low level transfers.
psp_ppp_ser_rx()	Receives data over the UART.
psp_ppp_ser_tx()	Sends data over the UART.
psp_ppp_ser_tx_ch()	Write one character over the UART.

These functions are described in the following sections.

psp_ppp_ser_init

Use this function to initialize the interface to the USB Host CDC-ACM module.

> **Note:** The USB host **usbh_cdcacm_init()** function must be called before this function.

Format

```
t_psp_ppp_ser_ret psp_ppp_ser_init (
    const uint8_t                   uid,
    const t_psp_ppp_ser_dsc * const   p_dsc )
```

Arguments

Argument	Description	Type
uid	The unit ID.	uint8_t
p_dsc	A pointer to the UART descriptor.	t_psp_ppp_ser_dsc *

Return values

Return value	Description
PSP_PPP_SER_SUCCESS	Successful execution.
PSP_PPP_SER_ERROR	Operation failed.

psp_ppp_ser_start

Use this function to start the USB Host CDC-ACM serial driver.

> **Note:** You must call **psp_ppp_ser_init()** before this function.

Format

```
t_psp_ppp_ser_ret psp_ppp_ser_start (
    const uint8_t           uid,
    const t_psp_ppp_ser_cb  cb,
    const uint32_t          cb_param )
```

Arguments

Argument	Description	Type
uid	The unit ID.	uint8_t
cb	The callback function.	t_psp_ppp_ser_cb
cb_param	The callback parameter.	uint32_t

Return values

Return value	Description
PSP_PPP_SER_SUCCESS	Successful execution.
PSP_PPP_SER_ERROR	Operation failed.

psp_ppp_ser_stop

Use this function to stop the USB Host CDC-ACM serial driver.

Format

```
t_psp_ppp_ser_ret psp_ppp_ser_stop( const uint8_t uid )
```

Arguments

Argument	Description	Type
uid	The unit ID.	uint8_t

Return values

Return value	Description
PSP_PPP_SER_SUCCESS	Successful execution.
PSP_PPP_SER_ERROR	Operation failed.

psp_ppp_ser_delete

Use this function to delete the USB Host CDC-ACM serial driver and release the associated resources.

Format

```
t_psp_ppp_ser_ret psp_ppp_ser_delete( const uint8_t uid )
```

Arguments

Argument	Description	Type
uid	The unit ID.	uint8_t

Return values

Return value	Description
PSP_PPP_SER_SUCCESS	Successful execution.
PSP_PPP_SER_ERROR	Operation failed.

ppp_ser_tx_advance

Use this function to advance a Tx transfer. This function is called by the PPP serial driver every time a Tx transfer is finished.

> **Note:** The caller must provide mutex protection.

Format

```
t_psp_ppp_ser_ret psp_ppp_ser_tx_advance( const uint8_t uid )
```

Arguments

Argument	Description	Type
uid	The unit ID.	uint8_t

Return values

Return value	Description
PSP_PPP_SER_SUCCESS	Successful execution.
PSP_PPP_SER_ERROR	Operation failed.

psp_ppp_ser_rx

Use this function to receive data over USB Host CDC-ACM.

Format

```
t_psp_ppp_ser_ret psp_ppp_ser_rx (
    const uint8_t      uid,
    uint8_t * const    p_buf,
    const uint32_t     buf_len,
    uint32_t * const   p_read_len )
```

Arguments

Argument	Description	Type
uid	The unit ID.	uint8_t
p_buf	A pointer to the buffer to read into.	uint8_t *
buf_len	The buffer length.	uint32_t
p_read_len	Where to write the number of bytes read.	uint32_t *

Return values

Return value	Description
PSP_PPP_SER_SUCCESS	Successful execution.
PSP_PPP_SER_ERROR	Operation failed.

psp_ppp_ser_tx

Use this function to send data over the USB Host CDC-ACM.

Format

```
t_psp_ppp_ser_ret psp_ppp_ser_tx (
    const uint8_t      uid,
    uint8_t * const    p_buf,
    const uint32_t     buf_len )
```

Arguments

Argument	Description	Type
uid	The unit ID.	uint8_t
p_buf	A pointer to the buffer to write from.	uint8_t *
buf_len	The buffer length.	uint32_t

Return values

Return value	Description
PSP_PPP_SER_SUCCESS	Successful execution.
PSP_PPP_SER_ERROR	Operation failed.

psp_ppp_ser_tx_ch

Use this function to write one character over the USB Host CDC-ACM.

Format

```
t_psp_ppp_ser_ret psp_ppp_ser_tx_ch (
    const uint8_t   uid,
    const uint8_t   ch )
```

Arguments

Argument	Description	Type
uid	The unit ID.	uint8_t
ch	The character to write.	uint8_t

Return values

Return value	Description
PSP_PPP_SER_SUCCESS	Successful execution.
PSP_PPP_SER_ERROR	Operation failed.