HTTPS Secure Client User Guide

Version 1.00

For use with HTTPS Secure Client module versions 1.02 and above

Date: 25-Apr-2017 16:10

All rights reserved. This document and the associated software are the sole property of HCC Embedded. Reproduction or duplication by any means of any portion of this document without the prior written consent of HCC Embedded is expressly forbidden.

HCC Embedded reserves the right to make changes to this document and to the related software at any time and without notice. The information in this document has been carefully checked for its accuracy; however, HCC Embedded makes no warranty relating to the correctness of this document.

Table of Contents

System Overview 4
- Introduction 4
- Methods 5
- Feature Check 5
- Packages and Documents 6
  - Packages 6
  - Documents 6

Source File List 7
- API Header File 7
- Configuration File 7
- System File 7
- Version File 7

Configuration Options 8

Application Programming Interface 10
- Module Management 10
  - httpc_init 11
  - httpc_start 12
  - httpc_stop 13
  - httpc_delete 14
- Client Management 15
  - httpc_start_request 16
  - httpc_register_cb 17
- Callback Functions 18
  - t_httpc_req_open_cb 19
  - t_httpc_req_read_cb 20
  - t_httpc_req_close_cb 21
  - t_httpc_request_hdr_cb 22
  - t_httpc_resp_open_cb 23
  - t_httpc_resp_write_cb 24
  - t_httpc_resp_close_cb 25
  - t_httpc_response_hdr_cb 26
  - t_httpc_next_request_cb 27
  - t_httpc_conn_closed_cb 28
- Error Codes 29
- Types and Definitions 30
  - t_httpc_cb_dsc 30
  - t_httpc_media_type 30
  - t_httpc_req_ctype 31
  - t_httpc_req_info 31
  - t_httpc_resp_info 31
  - t_httpc_method 32
  - t_httpc_close_ntf 32

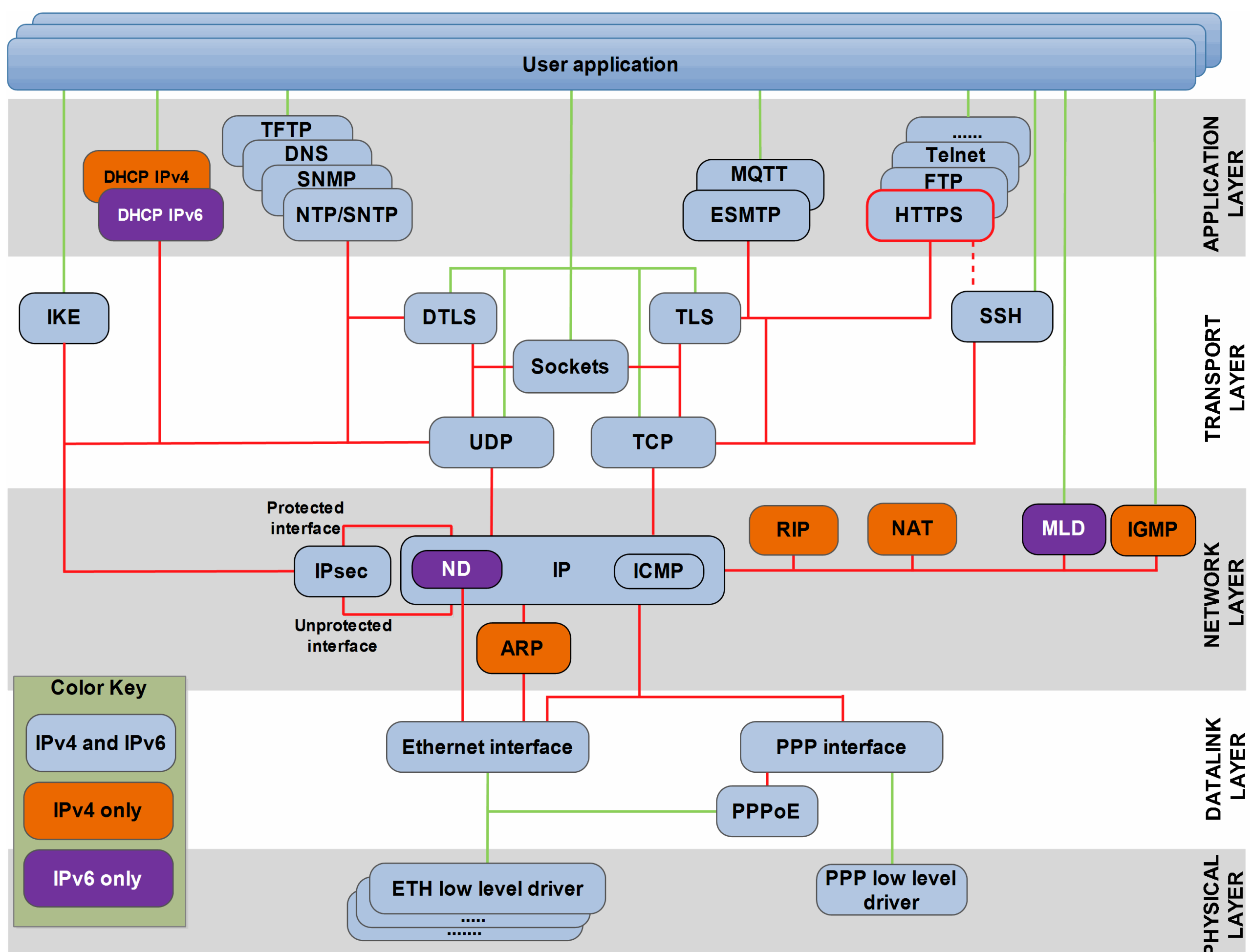
Integration 33
OS Abstraction Layer 33
Utilities 33
PSP Porting 34
Using the Demo Package 35
Demo Source Files 35
Demo Configuration Options 35
Demo Package API 36
httpc_user_init 36
httpc_user_start 37
httpc_user_stop 38
httpc_user_delete 39
Apache HTTP Server Setup 40
Testing the PUT and DELETE Methods 40
Testing the POST and GET Methods 41
Running the Tests 41

1 System Overview

1.1 Introduction

This guide is for those who want to implement an HTTPS secure client as part of HCC Embedded's MISRA-compliant TCP/IP stack. This HTTPS secure client provides an Application Programming Interface (API) for retrieving content from a web server. It supports the GET, HEAD, POST, PUT, and DELETE methods.

The HTTPS secure client module works with both IPv4 and IPv6 networks. Its place in the HCC TCP/IP stack is 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.)



The Hypertext Transfer Protocol Secure (HTTPS) provides secure communication over computer networks. HTTPS resources are identified and located on the network using Uniform Resource Identifiers (URIs). Where this manual refers to HTTP, the reference applies equally to HTTPS.

HTTPS operates as a request-response protocol in the client/server model. The HTTPS secure client provides an API for retrieving content from a web server.

The secure client sends an HTTPS request message to the server. The server, which may provide resources such as HTML files to the client, or perform other functions for it, sends a response message back to the client. This contains completion status information about the request. It may also contain requested content within its message body. The entire HTTPS message is encrypted, including the headers and the content.

1.2 Methods

This client module provides the following methods:

Method	Description
GET	Retrieves the information identified by the Request-URI.
HEAD	This is identical to GET except that the server returns only the metadata held in the HTTP headers. It does not return the message body itself. This is useful for obtaining metadata about an entity without transferring the body of the entity itself.
POST	Requests that the origin server accept the specified entity as a new subordinate of the resource identified by the Request-URI in the Request-Line. This method is used, for example, to post to a forum.
PUT	Requests that the enclosed entity be stored under the supplied Request-URI.
DELETE	Requests that the server delete the resource identified by the Request-URI.

1.3 Feature Check

The main features of the HTTPS Secure Client are the following:

- Conforms to the HCC Advanced Embedded Framework.
- Compliant with HCC's MISRA-compliant TCP/IP stack.
- Works with both IPv4 and IPv6 networks.
- Compliant with RFC 2818.
- Designed for integration with both RTOS and non-RTOS based systems.
- Can be configured to use BSD sockets.
- Supports all standard HTTP methods: GET, PUT, POST, and DELETE.
- Supports HTTP Secure (HTTPS) connections
- Handles a configurable number of simultaneous connections.
- Demo package provides sample implementations to base your HTTPS Secure Client on.

1.4 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_app_httpc	The HTTPS Secure Client package described in this manual.
ip_app_httpc_demo	The demo package of sample implementations that you can base your secure client on.
ip_app_http	The HTTPS Secure Server package.
mip_base	The TCP/IP Dual Stack base package.

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 HTTPS Secure Client User Guide

This is this document. This includes a section on how to use the demo package.

2 Source File List

The following sections describe 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_ip_app_httpc.h** is the only file that should be included by an application using this module. For details of these API functions, see Application Programming Interface.

2.2 Configuration File

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

2.3 System File

The system file is **src/ip/apps/http/httpc.c. This file should only be modified by HCC**.

2.4 Version File

The file **src/version/ver_ip_app_httpc.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_ip_app_httpc.h**. This section lists the options and their default values.

HTTPC_CONN_TASK_STACK_SIZE

The HTTP connection task stack size. The default value is 2048. All request (open, write, and close) and response (open, read, and close) functions are called from this task, so set the stack size accordingly.

HTTPC_SRV_PORT

The default server port. The default value is 80.

HTTPC_MAX_CONNECTIONS

The maximum number of connections. The default value is 4.

HTTPC_MAX_TRANSFER_UNIT

The maximum transfer size a connection can send at a time. The default value is 2048. Depending on the size of the assigned buffers, this can be slightly increased.

HTTPC_RESPONSE_TIMEOUT

The idle timeout. This is the time in seconds after which the connection is closed if there is no communication on it. The default value is 10.

Note: For this timeout the precision is 500ms, so timeout may occur 500ms earlier.

HTTPC_MAX_REQUEST_URI_BUF_SIZE

The maximum file request buffer size. The default value is 128. The request buffer holds the request URI, the optional URL-encoded INPUT string.

HTTPC_MAX_REQUEST_HOST_SIZE

The maximum request host buffer size. The default value is 64. The request buffer holds the host header value.

Note: The following two options define the default parameters written to the request header.

HTTPC_HEADER_VER

The default header HTML version. The default value is "HTTP/1.1".

HTTPC_HEADER_SERVER

The default header server name. The default value is "HCCWeb". This is one of the default parameters written to the request header.

HTTPC_USE_SOCKET

Keep the default of 0 to use the native HCC implementation. Set it to 1 to use the BSD socket implementation.

4 Application Programming Interface

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

4.1 Module Management

The functions are the following:

Function	Description
httpc_init()	Initializes the module and allocates the required resources.
httpc_start()	Starts the module.
httpc_stop()	Stops the module.
httpc_delete()	Deletes the module and releases the resources it used.

httpc_init

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

Note: Call this before any other function.

Format

```
t_httpc_ret httpc_init ( void )
```

Arguments

Argument
None.

Return Values

Return value	Description
HTTPC_SUCCESS	Successful execution.
HTTPC_ERROR	See Error Codes.

httpc_start

Use this function to start the HTTPS secure client module.

Note: Call **httpc_init()** before this function.

Format

```
t_httpc_ret httpc_start ( void )
```

Arguments

Argument
None.

Return Values

Return value	Description
HTTPC_SUCCESS	Successful execution.
HTTPC_ERROR	See Error Codes.

httpc_stop

Use this function to stop the HTTPS secure client module.

This stops ongoing requests immediately and closes TCP connections.

Format

```
t_httpc_ret httpc_stop ( void )
```

Arguments

Argument
None.

Return Values

Return value	Description
HTTPC_SUCCESS	Successful execution.
HTTPC_ERROR	See Error Codes.

httpc_delete

Use this function to delete the HTTPS secure client module and release the associated resources.

Format

```
t_httpc_ret httpc_delete ( void )
```

Arguments

Argument
None.

Return Values

Return value	Description
HTTPC_SUCCESS	Successful execution.
HTTPC_ERROR	See Error Codes.

4.2 Client Management

The functions are the following:

Function	Description
httpc_start_request()	Starts a request. This searches for a free connection and initiates a new connection to the HTTPS server.
http_register_cb()	Registers the callback functions.

httpc_start_request

Use this function to start a request.

This searches for a free connection and initiates a new connection to the HTTPS server.

Format

```
t_httpc_ret httpc_start_request (
    t_httpc_req_info *    p_req_info,
    uint16_t *            p_req_handler,
    uint16_t              conn_handler,
    uint16_t *            p_new_conn_handler )
```

Arguments

Name	Description	Type
p_req_info	A pointer to the HTTPS request information.	t_httpc_req_info *
p_req_handler	A pointer to the request handle.	uint16_t *
conn_handler	A pointer to an existing connection handle.	uint16_t
p_new_conn_handler	Where to write the new connection handle.	uint16_t *

Return Values

Code	Description
HTTPC_SUCCESS	Successful execution.
HTTPC_ERROR	Operation failed.

httpc_register_cb

Use this function to register callback functions.

Note: You must register the callback functions before starting a request.

Format

```
t_httpc_ret httpc_register_cb ( const t_httpc_cb_dsc * const p_cb_dsc )
```

Arguments

Name	Description	Type
p_cb_dsc	A pointer to the callback functions descriptor.	t_httpc_cb_dsc *

Return Values

Code	Description
HTTPC_SUCCESS	Successful execution.
HTTPC_ERROR	Operation failed.

4.3 Callback Functions

These functions are the interface between the HTTPS secure client and your file system. You can implement these as required.

Note:

- You must use **httpc_register_cb()** to register the callback functions before starting a request.
- All callback functions are called from the same HTTPS secure client task context and are protected against concurrent calls. That is, no callback function can interrupt another callback function.

Function	Description
t_httpc_req_open_cb()	Called before a PUT or POST request is sent.
t_httpc_req_read_cb()	Called to pass a data sent request to the server (with PUT or POST).
t_httpc_req_close_cb()	Called when all data has been sent by using PUT or POST.
t_httpc_request_hdr_cb()	Called to add headers before sending a request to the server.
t_httpc_resp_open_cb()	Called when a response is received.
t_httpc_resp_write_cb()	Called to pass data received in the response.
t_httpc_resp_close_cb()	Called when all response data has been received. This can be used to close a file or to release any other resource.
t_httpc_response_hdr_cb()	Called when a response is received.
t_httpc_next_request_cb()	Called when a new request can be sent on a keep-alive connection.
t_httpc_conn_closed_cb()	Called when a TCP connection is closed.

Copyright HCC Embedded 2017

t_httpc_req_open_cb

The **t_httpc_req_open_cb** definition specifies the format of the callback function that is called before a PUT or POST request is sent.

The user implementation (based on the request method) can decide whether a resource needs to be allocated (for example, a file in the PUT or POST cases). If the resource it tries to allocate is not currently available (for example, because there is no free file handle), it can signal that a later retry is required.

Format

```
typedef t_httpc_ret ( * t_httpc_req_open_cb ) (
    uint16_t           http_req_hdl,
    uint32_t * const   req_clen,
    uint32_t * const   p_hdl )
```

Arguments

Parameter	Description	Type
http_req_hdl	The handle of the current request.	uint16_t
req_clen	The length of the data to be sent.	uint32_t *
p_hdl	Where to write the internal handle. This handle is used by the **req_read()** and **req_close()** calls.	uint32_t *

Return Codes

Code	Description
HTTPC_SUCCESS	Successful execution.
HTTPC_RETRY	The client must retry later.
HTTPC_ERROR	Operation failed.

t_httpc_req_read_cb

The **t_httpc_req_read_cb** definition specifies the format of the callback function that is called to pass a data sent request to the server (with PUT or POST).

> **Note:** This function is always called from the same task context as **t_httpc_req_open_cb()**.

Format

```
t_httpc_ret httpc_req_read_cb (
    uint16_t            http_req_hdl,
    const uint32_t      hdl,
    uint8_t * const     p_buf,
    const uint16_t      buf_len,
    uint16_t * const    p_rd_len )
```

Arguments

Parameter	Description	Type
http_req_hdl	The handle of the current request.	uint16_t
hdl	The internal handle returned by **t_httpc_req_open_cb()**.	uint32_t
p_buf	Where to write the request content data.	uint8_t *
buf_len	The length of the output buffer.	uint16_t
p_rd_len	Where to write the length of the request content data.	uint16_t *

Return Codes

Code	Description
HTTPC_SUCCESS	Successful execution.
HTTPC_ERROR	Operation failed.

t_httpc_req_close_cb

The **t_httpc_req_close_cb** definition specifies the format of the callback function that is called when all data has been sent by using PUT or POST.

This callback can be used to close a file or to free any other resource.

> **Note:** This function is always called from the same task context as **t_httpc_req_open_cb()**.

Format

```
typedef t_httpc_ret ( * t_httpc_req_close_cb ) (
    uint16_t         http_req_hdl,
    const uint32_t   hdl )
```

Arguments

Parameter	Description	Type
http_req_hdl	The handle of the current request.	uint16_t
hdl	The internal handle returned by **t_httpc_req_open_cb()**.	uint32_t

Return Codes

Code	Description
HTTPC_SUCCESS	Successful execution.
HTTPC_ERR_NOT_FOUND	No response entity was found.
HTTPC_ERROR	Operation failed.

t_httpc_request_hdr_cb

The **t_httpc_request_hdr_cb** definition specifies the format of the callback function that is called to add headers before sending a request to the server.

Note:

- The Method line, host, and user-agent must already be set in the buffer with the values specified in **httpc_start_request()** input parameter.
- If specified, the content length is added by the client after calling the **httpc_req_open_cb()** callback.

Format

```
typedef t_httpc_ret ( * t_httpc_request_hdr_cb )(
    uint16_t           http_req_hdl,
    char_t * const     p_headers,
    uint16_t const     max_length,
    uint16_t * const   p_length_added )
```

Arguments

Parameter	Description	Type
http_req_hdl	The handle of the current request.	uint16_t
p_headers	A pointer to the buffer that contains the headers. This points to the position where the next header can be added.	char_t *
max_length	The length of the header buffer.	uint16_t
p_length_added	The length of the headers added to the buffer.	uint16_t *

Return Codes

Code	Description
HTTPC_SUCCESS	Successful execution.
HTTPC_ERROR	Operation failed.

Copyright HCC Embedded 2017

t_httpc_resp_open_cb

The **t_httpc_resp_open_cb** definition specifies the format of the callback function that is called when a response is received.

The user implementation (based on the request method) can decide whether a resource needs to be allocated. If it tries to do this and the resource is not currently available (for example, because there is no free file handle), it can signal that a later retry is needed. This function can be called to any kind of method that has a content greater than zero.

> **Note:** Always call this function from the same task context as **t_httpc_req_open_cb()**.

Format

```
typedef t_httpc_ret ( * t_httpc_resp_open_cb ) (
    uint16_t          http_req_hdl,
    uint32_t * const  p_new_hdl )
```

Arguments

Parameter	Description	Type
http_req_hdl	The handle of the current request returned by **t_httpc_req_open_cb()**.	uint16_t
p_new_hdl	Where to write the new handle. This handle is used by the **resp_write()** and **resp_close()** calls.	uint32_t *

Return Codes

Code	Description
HTTPC_SUCCESS	Successful execution.
HTTPC_RETRY	The client must retry later.
HTTPC_ERROR	Operation failed.

t_httpc_resp_write_cb

The **t_httpc_resp_write_cb** definition specifies the format of the callback function that is called to pass data received in the response.

The implementation can decide how to process the data (for example, to store it in a structure or a file).

> **Note:** This function is always called from the same task context as **t_httpc_resp_open_cb()**.

Format

```
typedef t_httpc_ret ( * t_httpc_resp_write_cb ) (
    uint16_t          http_req_hdl,
    const uint32_t    hdl,
    uint8_t * const   p_buf,
    const uint16_t    buf_len )
```

Arguments

Parameter	Description	Type
http_req_hdl	The handle of the current request.	uint16_t
hdl	The internal handle returned by **t_httpc_resp_open_cb()**.	uint32_t
p_buf	Where to write the data received.	uint8_t *
buf_len	The length of the data received.	uint16_t

Return Codes

Code	Description
HTTPC_SUCCESS	Successful execution.
HTTPC_ERROR	Operation failed.

t_httpc_resp_close_cb

The **t_httpc_resp_close_cb** definition specifies the format of the callback function that is called when all response data has been received. This can be used to close a file or to release any other resource.

Note: This function is always called from the same task context as **t_httpc_resp_open_cb()**.

Format

```
typedef t_httpc_ret ( * t_httpc_resp_close_cb ) (
    uint16_t         http_req_hdl,
    const uint32_t   hdl )
```

Arguments

Parameter	Description	Type
http_req_hdl	The handle of the current request.	uint16_t
hdl	The internal handle returned by **t_httpc_resp_open_cb()**.	uint32_t

Return Codes

Code	Description
HTTPC_SUCCESS	Successful execution.
HTTPC_ERROR	Operation failed.

t_httpc_response_hdr_cb

The **t_httpc_response_hdr_cb** definition specifies the format of the callback function that is called when a response was received.

Format

```
typedef t_httpc_ret ( * t_httpc_response_hdr_cb )(
    uint16_t                    http_req_hdl,
    const char_t * const        p_headers,
    const t_httpc_resp_info *   p_resp_info )
```

Arguments

Parameter	Description	Type
http_req_hdl	The handle of the current request.	uint16_t
p_headers	A pointer to the buffer that contains the response headers.	char_t *
p_resp_info	A pointer to the response information structure.	t_httpc_resp_info *

Return Codes

Code	Description
HTTPC_SUCCESS	Successful execution.
HTTPC_ERROR	Operation failed.

t_httpc_next_request_cb

The **t_httpc_next_request_cb** definition specifies the format of the callback function that is called when a new request can be sent on a keep-alive connection.

Format

```
typedef t_httpc_ret ( * t_httpc_next_request_cb )(
    uint16_t   http_req_hdl,
    uint16_t   http_conn_hdl )
```

Arguments

Parameter	Description	Type
http_req_hdl	The handle of the current request.	uint16_t
http_conn_hdl	The handle of the connection that was closed.	uint16_t

Return Codes

Code	Description
HTTPC_SUCCESS	Successful execution.
HTTPC_ERROR	Operation failed.

t_httpc_conn_closed_cb

The **t_httpc_conn_closed_cb** definition specifies the format of the callback function that is called when a TCP connection is closed.

Format

```
typedef t_httpc_ret ( * t_httpc_conn_closed_cb )(
    uint16_t                 http_conn_hdl,
    const t_httpc_close_ntf   ntf )
```

Arguments

Parameter	Description	Type
http_conn_hdl	The handle of the connection that was closed.	uint16_t
ntf	The notification showing the reason that the connection was closed.	t_httpc_close_ntf

Return Codes

Code	Description
HTTPC_SUCCESS	Successful execution.
HTTPC_ERROR	Operation failed.

4.4 Error Codes

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

> **Note:** Check other error code values in the base system by using the *HCC TCP/IP Dual Stack System User Guide*.

Return Value	Value	Description
HTTPC_SUCCESS	0	Successful execution.
HTTPC_RETRY	1	Retry later.
HTTPC_ERR_CONN_BUSY	2	The connection is busy; no new requests are possible on it.
HTTPC_ERR_RESOURCE	3	No free HTTP connection is available.
HTTPC_ERR_PARAM	4	Parameter error.
HTTPC_ERROR	5	General error.
HTTPC_CONN_HDL_INVALID	0	The connection handle is invalid.

4.5 Types and Definitions

t_httpc_cb_dsc

The *t_httpc_cb_dsc* structure is the callback functions descriptor.

Element	Type	Description
t_httpc_req_open_cb	hcd_req_open_cb	The request open callback.
t_httpc_req_read_cb	hcd_req_read_cb	The request read callback.
t_httpc_req_close_cb	hcd_req_close_cb	The request close callback.
t_httpc_resp_open_cb	hcd_resp_open_cb	The response open callback.
t_httpc_resp_write_cb	hcd_resp_write_cb	The response write callback.
t_httpc_resp_close_cb	hcd_resp_close_cb	The response close callback.
t_httpc_request_hdr_cb	hcd_request_hdr_cb	The request header callback.
t_httpc_response_hdr_cb	hcd_response_hdr_cb	The response header callback.
t_httpc_next_request_cb	hcd_next_request_cb	The next request callback.
t_httpc_conn_closed_cb	hcd_conn_closed_cb	The connection closed callback.

t_httpc_media_type

The *t_httpc_media_type* structure holds the media type features. It takes this form:

Element	Type	Description
p_hme_type	char_t *	The content type.
p_hme_ext	char_t *	The file name extension.

t_httpc_req_ctype

The *t_httpc_req_ctype* typedef holds the request content types. These are as follows:

Method	Description
HTTPC_REQ_CTYPE_INVALID	Invalid type.
HTTPC_REQ_CTYPE_URLENCODED	URL encoded.
HTTPC_REQ_CTYPE_MULTIPART	Multipart or form data.
HTTPC_REQ_CTYPE_TEXT	Text/plain.

t_httpc_req_info

The *t_httpc_req_info* structure holds the client request information. It takes this form:

Element	Type	Description
server_ip_addr	t_ip_port	The server IP address and port.
hreq_method	t_httpc_method	The request method (see HTTPC_METHOD_xxx).
p_hreq_uri	char_t *	A pointer to the request URI.
p_host	char_t *	A pointer to the host.

t_httpc_resp_info

The *t_httpc_resp_info* structure holds the client response information. It takes this form:

Element	Type	Description
connection	t_httpc_connection	The connection type, keep-alive or close.
transfer_enc	t_transfer_encoding	The transfer encoding (chunked).
clen	uint32_t	The content length.

t_httpc_method

The *t_httpc_method* typedef specifies the available methods.

Method	Description
HTTPC_METHOD_INVALID	Invalid method.
HTTPC_METHOD_GET	Retrieves the data that is identified by the URI.
HTTPC_METHOD_HEAD	This method is similar to GET but returns just HTTP headers, not the document body.
HTTPC_METHOD_PUT	Specifies that the data in the body section must be stored under the supplied URL, which must already exist.
HTTPC_METHOD_POST	Creates a new object linked to the specified object.
HTTPC_METHOD_DELETE	Asks the server to delete the information corresponding to the given URL.

t_httpc_close_ntf

The *t_httpc_close_ntf* typedef gives the reason why a connection was closed, as follows:

Element	Description
HTTPC_NTF_NO_CONNECTION	No TCP connection is available.
HTTPC_NTF_CLOSED	The connection is closed.
HTTPC_NTF_TIMEOUT	The connection timed out.

5 Integration

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

5.1 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

5.2 Utilities

The HTTP 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.

5.3 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 *HCC Base Platform Support Package User Guide*.

The module makes use of the following standard PSP functions:

Function	Package	Element	Description
psp_strncat()	psp_base	psp_string	Appends a string.
psp_strncpy()	psp_base	psp_string	Copies one string of defined length to another.
psp_strnlen()	psp_base	psp_string	Gets the length of a string.

The module makes use of the following standard PSP macro:

Macro	Package	Element	Description
PSP_RD_BE32	psp_base	psp_endianness	Reads a 32 bit value stored as big-endian from a memory location.

6 Using the Demo Package

This module demonstrates the HTTPS secure client module. The supported methods are GET, HEAD, POST, PUT and DELETE.

The test works using memory buffers (**httpc_user.c**) or a file system (**httpc_user_fs.c**) as well. One source file is used at a time.

If the file system is used, the file **upl1.html** is created on the file system's root directory. This file is uploaded to the server using PUT; its content does not matter.

6.1 Demo Source Files

There are three files in the **demo** directory:

File	Description
httpc_user.c	Source code of the API functions.
httpc_user.h	Header file for the API functions.
httpc_user_fs.c	Demo configuration options and other elements.

6.2 Demo Configuration Options

Set the demo configuration options in the file **httpc_user_fs.c**. This section lists the options and their default values.

HTTP_SERVER_IP_ADDRESS

The server IP address in hexadecimal format. The default is 192.168.11.1.

KEEP_ALIVE

Keep the default of 1 to use the same connection for all tests. Set the value to 0 to use a separate connection for each.

HTTPCT_USE_PRINTOUT

Keep the default of 1 to print debug messages.

Note: The host string *host_str[]* is also defined in this file. It is "www.hcctest.com" by default.

6.3 Demo Package API

There are just four functions in the demo.

httpc_user_init

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

Note: Call this before any other function.

Format

```
t_httpc_ret httpc_user_init ( void )
```

Arguments

Argument
None.

Return Values

Return value	Description
HTTPC_SUCCESS	Successful execution.
HTTPC_ERROR	See Error Codes.

httpc_user_start

Use this function to start the secure client demo module. This starts the sequence of HTTP requests to run the tests.

> **Note:** Call **httpc_user_init()** before this function.

Format

```
t_httpc_ret httpc_user_start ( void )
```

Arguments

Argument
None.

Return Values

Return value	Description
HTTPC_SUCCESS	Successful execution.
HTTPC_ERROR	See Error Codes.

httpc_user_stop

This function has no effect currently.

Format

```
t_httpc_ret httpc_user_stop ( void )
```

Arguments

Argument
None.

Return Values

Return value	Description
HTTPC_SUCCESS	Successful execution.
HTTPC_ERROR	See Error Codes.

httpc_user_delete

Use this function to delete the secure client demo module and release the associated resources.

Format

```
t_httpc_ret httpc_user_delete ( void )
```

Arguments

Argument
None.

Return Values

Return value	Description
HTTPC_SUCCESS	Successful execution.
HTTPC_ERROR	See Error Codes.

6.4 Apache HTTP Server Setup

Testing the PUT and DELETE Methods

To test PUT and DELETE, install an Apache HTTP server on a (virtual) machine. Install and enable the WebDAV module (**mod_dav**).

Note: The following configuration enables uploading of files to the server, so it is not secure.

Configure Apache version 2.4.7-1ubuntu4 on the machine as follows:

```
/etc/apache2/apache2.conf:
<Directory /var/www/html>
    Options Indexes FollowSymLinks
    AllowOverride None
    Require all granted
    Dav On
    <Limit GET POST PUT DELETE HEAD OPTIONS>
        Order allow,deny
        Allow from all
    </Limit>
    <LimitExcept GET POST PUT DELETE HEAD OPTIONS>
        Order deny,allow
        Deny from all
    </LimitExcept>
</Directory>
```

Testing the POST and GET Methods

To test POST and GET, install a CGI script named **cgi_test.py** in the Apache server's html directory. The content of the file **cgi_test.py** is as follows:

```
#!/usr/bin/python
import cgi, cgitb
form = cgi.FieldStorage()
first_name = form.getvalue('first_name')
last_name  = form.getvalue('last_name')
print "Content-type: text/html\r\n\r\n"
print "<html><head><title>CGI Program</title></head>"
print "<body><h2>Hello %s %s</h2></body></html>" % (first_name, last_name)
```

6.5 Running the Tests

Start the tests by calling **http_user_start()**. The following test cases will run:

Method	File used	Description
1 PUT	upl1.html	Uploads the file **upl1.html** to the HTTP server.
2 HEAD	upl1.html	Gets information about the uploaded file **upl1.html**.
3 GET	upl1.html	Downloads the file **upl1.html** and saves it to **get1.html**.
4 GET	cgi_test.py	Tests server side scripts (also known as CGI) using GET. The server's answer is saved to the file **get2.html**.
5 POST	cgi_test.py	Tests server side scripts (also known as CGI) using POST. The server's answer is saved to the file **post1.html**.
6 DELETE	upl1.html	Deletes the file **upl1.html** from the HTTP server.