



# HTTP Client User Guide

Version 2.00

For use with HTTP Client module versions 3.01 and  
above

## Table of Contents

- 1. System Overview** ..... 4
  - 1.1. Introduction** ..... 5
  - 1.2. Methods** ..... 6
  - 1.3. Feature Check** ..... 7
  - 1.4. Packages and Documents** ..... 8
  - 1.5. Change History** ..... 9
- 2. Source File List** ..... 10
- 3. Configuration Options** ..... 11
- 4. Application Programming Interface** ..... 13
  - 4.1. Module Management** ..... 13
    - httpc\_init ..... 14
    - httpc\_start ..... 15
    - httpc\_stop ..... 16
    - httpc\_delete ..... 17
  - 4.2. Client Management** ..... 18
    - httpc\_start\_request ..... 19
    - httpc\_register\_cb ..... 20
  - 4.3. Callback Functions** ..... 21
    - t\_httpc\_req\_open\_cb ..... 22
    - t\_httpc\_req\_read\_cb ..... 23
    - t\_httpc\_req\_close\_cb ..... 24
    - t\_httpc\_request\_hdr\_cb ..... 25
    - t\_httpc\_resp\_open\_cb ..... 26
    - t\_httpc\_resp\_write\_cb ..... 27
    - t\_httpc\_resp\_close\_cb ..... 28
    - t\_httpc\_response\_hdr\_cb ..... 29
    - t\_httpc\_next\_request\_cb ..... 30
    - t\_httpc\_conn\_closed\_cb ..... 31
  - 4.4. Error Codes** ..... 32
  - 4.5. Types and Definitions** ..... 33
    - t\_httpc\_cb\_dsc ..... 33
    - t\_httpc\_media\_type ..... 33
    - t\_httpc\_req\_ctype ..... 34
    - t\_httpc\_req\_info ..... 34

t_httpc_resp_info .....	34
t_httpc_method .....	35
t_httpc_close_ntf .....	35
<b>5. Integration .....</b>	<b>36</b>
<b>5.1. OS Abstraction Layer .....</b>	<b>36</b>
<b>5.2. Utilities .....</b>	<b>36</b>
<b>5.3. PSP Porting .....</b>	<b>37</b>
<b>6. Using the Demo Package .....</b>	<b>38</b>
<b>6.1. Demo Source Files .....</b>	<b>38</b>
<b>6.2. Demo Configuration Options .....</b>	<b>38</b>
<b>6.3. Demo Package API .....</b>	<b>39</b>
httpc_user_init .....	40
httpc_user_start .....	41
httpc_user_stop .....	42
httpc_user_delete .....	43
<b>6.4. Apache HTTP Server Setup .....</b>	<b>44</b>
<b>6.5. Running the Tests .....</b>	<b>45</b>
<b>7. Version .....</b>	<b>46</b>

# 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. This section includes a diagram showing the position of this module within HCC's TCP/IP stack.
- [Methods](#) - summarizes the methods provided by the module: GET, HEAD, PUT, POST and DELETE.
- [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.

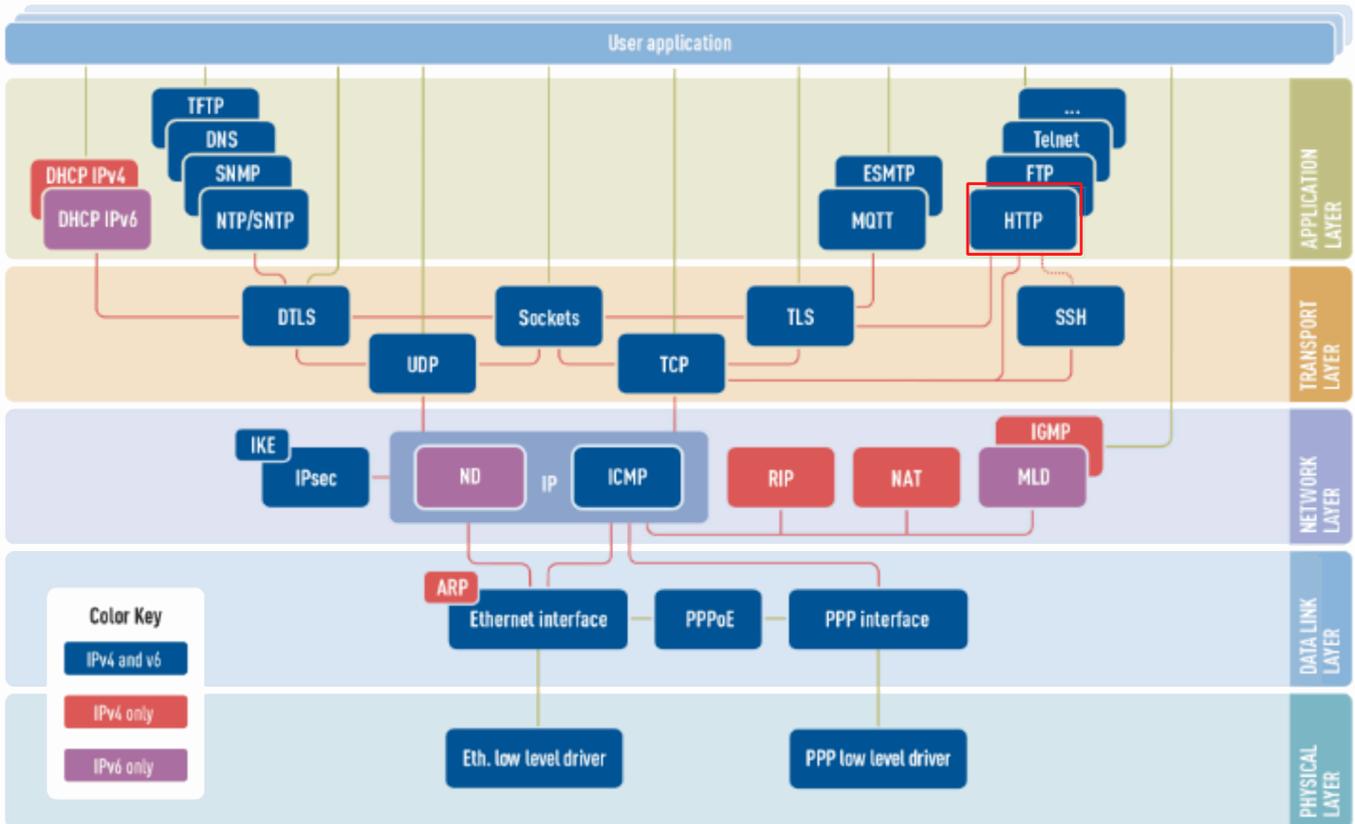
All rights reserved. This document and the associated software are the sole property of HCC Embedded. Reproduction or duplication by any means of any portion of this document without the prior written consent of HCC Embedded is expressly forbidden.

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.

# 1.1. Introduction

This guide is for those who want to implement an HTTP client as part of HCC Embedded’s MISRA-compliant TCP/IP stack. This HTTP 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 HTTP 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 (HTTP) and its secure equivalent (HTTPS) provide communication over computer networks. HTTP resources are identified and located on the network using Uniform Resource Identifiers (URIs).

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

The client sends an HTTP 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.

## 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 HTTP 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 connections.
- Supports HTTP Secure connections.
- Handles a configurable number of simultaneous connections.
- Demo package provides sample implementations to base your HTTP 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
<b>hcc_base_doc</b>	This contains the two guides that will help you get started.
<b>ip_app_httpc</b>	The HTTP Client package described in this manual.
<b>ip_app_httpc_demo</b>	The demo package of sample implementations that you can base your client on.
<b>mip_base</b>	The TCP/IP Dual Stack base package.
<b>mip_tcp</b>	The TCP package.
<b>ip_tls</b>	Needed if TLS/DTLS support is required.

### 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 HTTP Client User Guide

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

## 1.5. 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\\_app\\_httpc](#) or [History: ip\\_app\\_httpc\\_demo](#).

The current version of this manual is 2.00. The full list of versions is as follows:

Manual version	Date	Software version	Reason for change
2.00	2020-02-20	3.01	New document template.
1.40	2019-02-15	3.01	Added <b>ip_tls</b> to Packages list. Modified <b>httpc_start_request()</b> .
1.30	2018-10-04	2.04	Added <i>secure</i> to <i>t_httpc_req_info</i> .
1.20	2018-02-09	2.03	Added HTTPS configuration options for secure connections.
1.10	2017-07-19	2.02	Changed title from HTTPS Secure Client to HTTP Client. New <i>Change History</i> format. Introduced <i>t_httpc_user_hdl</i> in callbacks.
1.00	2017-04-25	1.02	First online version.

## 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.

### 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](#).

### 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](#).

### System File

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

### 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.

### Demo Package

The package **ip\_app\_httpc\_demo** holds these files in **psp/board/demo**:

File	Description
<b>httpc_user.h</b>	Header file defining the <b>httpc_user_init/start/stop/delete()</b> functions for the demo.
<b>httpc_user.c</b>	Implements HTTP Client callback functions and demonstrates GET/POST/PUT/DELETE methods.
<b>http_user_fs.c</b>	Implements HTTP Client callback functions, demonstrating the methods interfaced to a FAT file system.

The demo also has its own version file, **ver\_ip\_app\_httpc\_user.h**.

## 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\_USER\_AGENT**

The default user agent name. The default value is "HCC demo".

## HTTPC\_USE\_SOCKET

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

**Note:** The following two options configure HTTP Secure connections.

## HTTPC\_SECURE\_ENABLE

Keep the default of 1 to enable HTTP Secure connections, otherwise set it to 0. **This option must be enabled for dynamic variables.**

## HTTPC\_SEC\_TRANSFER\_UNIT

The HTTP secure connection transfer unit for socket connection. The default value is 1200.

This must be less than the maximum data that can be transferred by TLS in a single secure PDU.

## 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
<b>httpc_init()</b>	Initializes the module and allocates the required resources.
<b>httpc_start()</b>	Starts the module.
<b>httpc_stop()</b>	Stops the module.
<b>httpc_delete()</b>	Deletes the module and releases the resources it used.

## httpc\_init

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

**Note:** Call this before any other function.

### Format

```
t_httpc_ret httpc_init ( void )
```

### Arguments

None.

### Return Values

Return value	Description
HTTPC_SUCCESS	Successful execution.
HTTPC_ERROR	See <a href="#">Error Codes</a> .

## httpc\_start

Use this function to start the client module.

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

### Format

```
t_httpc_ret httpc_start ( void )
```

### Arguments

None.

### Return Values

Return value	Description
HTTPC_SUCCESS	Successful execution.
HTTPC_ERROR	See <a href="#">Error Codes</a> .

## httpc\_stop

Use this function to stop the client module.

This stops ongoing requests immediately and closes TCP connections.

### Format

```
t_httpc_ret httpc_stop ( void )
```

### Arguments

None.

### Return Values

Return value	Description
HTTPC_SUCCESS	Successful execution.
HTTPC_ERROR	See <a href="#">Error Codes</a> .

## httpc\_delete

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

### Format

```
t_httpc_ret httpc_delete ( void )
```

### Arguments

None.

### Return Values

Return value	Description
HTTPC_SUCCESS	Successful execution.
HTTPC_ERROR	See <a href="#">Error Codes</a> .

## 4.2. Client Management

The functions are the following:

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

## httpc\_start\_request

Use this function to start a request. It works in one of two ways:

- If *conn\_handle* is HTTPC\_CONN\_HDL\_INVALID, it searches for a free connection and initiates a new connection to the HTTP server.
- Otherwise it uses the existing connection and executes the request.

Note: Before using this function, register callback functions by using **httpc\_register\_cb()**.

### Format

```
t_httpc_ret httpc_start_request (
    t_httpc_req_info * p_req_info,
    uint16_t          conn_handle,
    uint16_t *       p_new_conn_handle,
    uint16_t *       p_req_handle )
```

### Arguments

Name	Description	Type
p_req_info	A pointer to the HTTP request information.	<a href="#">t_httpc_req_info</a> *
conn_handle	A pointer to an existing connection handle or, for a new connection, HTTPC_CONN_HDL_INVALID.	uint16_t
p_new_conn_handle	If <i>conn_handle</i> is HTTPC_CONN_HDL_INVALID this returns a pointer to the new connection handle.	uint16_t*
p_req_handle	On return, a pointer to the request handle. This handle is used for all callbacks.	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.	<a href="#">t_httpc_cb_dsc</a> *

### Return Values

Code	Description
HTTPC_SUCCESS	Successful execution.
HTTPC_ERROR	Operation failed.

## 4.3. Callback Functions

These functions are the interface between the HTTP 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 HTTP client task context and are protected against concurrent calls. That is, no callback function can interrupt another callback function.

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

## 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,
    t_httpc_user_hdl * 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 <b>req_read()</b> and <b>req_close()</b> calls.	t_httpc_user_hdl *

### 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 t_httpc_user_hdl 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 <b>t_httpc_req_open_cb()</b> .	t_httpc_user_hdl
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 t_httpc_user_hdl  hdl )
```

### Arguments

Parameter	Description	Type
http_req_hdl	The handle of the current request.	uint16_t
hdl	The internal handle returned by <b>t_httpc_req_open_cb()</b> .	t_httpc_user_hdl

### 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.

## 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,
    t_httpc_user_hdl * const  p_new_hdl )
```

### Arguments

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

### 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 t_httpc_user_hdl  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 <b>t_httpc_resp_open_cb()</b> .	t_httpc_user_hdl
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 t_httpc_user_hdl  hdl )
```

### Arguments

Parameter	Description	Type
http_req_hdl	The handle of the current request.	uint16_t
hdl	The internal handle returned by <b>t_httpc_resp_open_cb()</b> .	t_httpc_user_hdl

### 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.	<a href="#">t_httpc_resp_info</a> *

### 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.	<a href="#">t_httpc_close_ntf</a>

### 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_REMOTE_CLOSED	6	The remote side closed the connection.

HTTPC\_CONN\_HDL\_INVALID indicates that the connection handle is invalid.

## 4.5. Types and Definitions

This section describes the main elements that are defined in the API Header file.

### **t\_httpc\_cb\_dsc**

The *t\_httpc\_cb\_dsc* structure is the callback functions descriptor.

Element	Type	Description
hcd_req_open_cb	t_httpc_req_open_cb	The request open callback.
hcd_req_read_cb	t_httpc_req_read_cb	The request read callback.
hcd_req_close_cb	t_httpc_req_close_cb	The request close callback.
hcd_resp_open_cb	t_httpc_resp_open_cb	The response open callback.
hcd_resp_write_cb	t_httpc_resp_write_cb	The response write callback.
hcd_resp_close_cb	t_httpc_resp_close_cb	The response close callback.
hcd_request_hdr_cb	t_httpc_request_hdr_cb	The request header callback.
hcd_response_hdr_cb	t_httpc_response_hdr_cb	The response header callback.
hcd_next_request_cb	t_httpc_next_request_cb	The next request callback.
hcd_conn_closed_cb	t_httpc_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	x-www-form-urlencoded.
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.
secure	uint8_t	Flag indicating a secure connection. (Only if configuration option <a href="#">HTTPC_SECURE_ENABLE</a> is set.)
hreq_method	t_httpc_method	The request method (see <a href="#">HTTPC_METHOD_xxx</a> ).
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_HEAD	This method is similar to GET but returns just HTTP headers, not the document body.
HTTPC_METHOD_GET	Retrieves the data that is identified by the URI.
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

The demo module uses the following OAL components:

OAL Resource	Number Required
Tasks	1
Mutexes	0
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
<b>psp_strncat()</b>	psp_base	psp_string	Appends a string.
<b>psp_strncpy()</b>	psp_base	psp_string	Copies one string of defined length to another.
<b>psp_strlen()</b>	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 HTTP 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
<b>httpc_user.c</b>	Source code of the API functions.
<b>httpc_user.h</b>	Header file for the API functions.
<b>httpc_user_fs.c</b>	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](http://www.hcctest.com)" by default.

## 6.3. Demo Package API

The functions in the demo are the following:

Function	Description
<b>httpc_user_init()</b>	Initializes the client demo module and allocates the required resources.
<b>httpc_user_start()</b>	Starts the client demo module.
<b>httpc_user_stop()</b>	Stops the client demo module.
<b>httpc_user_delete()</b>	Deletes the client demo module and releases the resources it used.

## httpc\_user\_init

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

**Note:** Call this before any other function.

### Format

```
t_httpc_ret httpc_user_init ( void )
```

### Arguments

None.

### Return Values

Return value	Description
HTTPC_SUCCESS	Successful execution.
HTTPC_ERROR	See <a href="#">Error Codes</a> .

## httpc\_user\_start

Use this function to start the 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

None.

### Return Values

Return value	Description
HTTPC_SUCCESS	Successful execution.
HTTPC_ERROR	See <a href="#">Error Codes</a> .

## httpc\_user\_stop

This function has no effect currently.

### Format

```
t_httpc_ret httpc_user_stop ( void )
```

### Arguments

None.

### Return Values

Return value	Description
HTTPC_SUCCESS	Successful execution.
HTTPC_ERROR	See <a href="#">Error Codes</a> .

## httpc\_user\_delete

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

### Format

```
t_httpc_ret httpc_user_delete ( void )
```

### Arguments

None.

### Return Values

Return value	Description
HTTPC_SUCCESS	Successful execution.
HTTPC_ERROR	See <a href="#">Error Codes</a> .

## 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('&#39;first_name&#39;')
last_name = form.getvalue('&#39;last_name&#39;')
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 <b>upl1.html</b> to the HTTP server.
2 HEAD	upl1.html	Gets information about the uploaded file <b>upl1.html</b> .
3 GET	upl1.html	Downloads the file <b>upl1.html</b> and saves it to <b>get1.html</b> .
4 GET	cgi_test.py	Tests server side scripts (also known as CGI) using GET. The server's answer is saved to the file <b>get2.html</b> .
5 POST	cgi_test.py	Tests server side scripts (also known as CGI) using POST. The server's answer is saved to the file <b>post1.html</b> .
6 DELETE	upl1.html	Deletes the file <b>upl1.html</b> from the HTTP server.

## 7. Version

Version 2.00

For use with HTTP Client module versions 3.01 and above